

UNIVERSITÀ DEGLI STUDI DI MILANO

DIPARTIMENTO DI SCIENZE DELL'INFORMAZIONE

Generazione casuale e
conteggio approssimato
di strutture combinatorie

*Random Generation and
Approximate Counting
of Combinatorial Structures*

Tesi di Dottorato di Massimo Santini

Relatori Prof. Alberto Bertoni
Prof. Massimiliano Goldwurm

Relatore esterno Prof. Bruno Codenotti

DOTTORATO IN SCIENZE DELL'INFORMAZIONE — XI CICLO

Massimo Santini c/o
Dipartimento di Scienze dell'Informazione
Università Statale degli Studi di Milano
Via Comelico 39/41, 20135 Milano — Italia
e-mail santini@dsi.unimi.it

Ad Ilenia, la persona che più di tutte
ha cambiato la mia vita.

Voglio ringraziare *Alberto Bertoni*, per il sostegno e l'ispirazione che non mi ha mai fatto mancare, dal periodo della mia laurea ad oggi. *Paola Campadelli*, che mi ha guidato in questi miei primi anni di ricerca, aiutandomi a maturare ed addolcire il mio carattere così bellicoso e critico. *Massimiliano Goldwurm*, per la pazienza e l'attenzione infinita che ha dedicato a questo lavoro, mostrandomi sempre rispetto ed amicizia.

Ringrazio anche *Giuliano Grossi*, per le sue competenze ed il materiale che mi ha messo a disposizione in questi mesi di lavoro, in particolare riguardo al capitolo 6. *Giovanni Pighizzini*, per avermi offerto il suo tempo e per avermi reso partecipe di alcuni suoi interessanti risultati, in particolare riguardo alla sezione 4.2.

Ringrazio infine *Paolo Boldi* e *Sebastiano Vigna*, per l'affetto che mi hanno dimostrato negli anni, i consigli ed il supporto che non mi hanno mai lesinato. Tutti i *colleghi del dottorato*, per aver costituito attorno a me, sin dall'inizio, un ambiente accogliente e stimolante per la ricerca. *Ilenia Epifani* (ultima, non certo per importanza), per essermi stata sempre a fianco, tollerando l'ansia di questi ultimi giorni di stesura e mai negandomi la sua revisione critica e obiettiva del mio lavoro.

INTRODUCTION

Combinatorial counting problems have a long and distinguished history. Apart from their intrinsic interest, they arise naturally from investigations in numerous branches of mathematics and natural sciences and have given rise to a rich and beautiful theory. Ranking problems, which consist in determining the position of a given element in a well-ordered set, are closely related to counting. Random generation problems are less well studied but have a large number of computational applications.

From the structural complexity viewpoint, the study of counting problems was initiated by Valiant (1979). A parallel approach to random generation problems was proposed by Jerrum, Valiant, and Vazirani (1986); in particular, they show how the standard reduction from generation to exact counting can be modified to yield an almost uniform generator giving only approximate counting estimates. They also locate the almost uniform generation and approximate counting problems for general NP relations¹ within the second level of the (probabilistic) polynomial time hierarchy (Stockmeyer, 1977). Finally, ranking has been studied by Huynh (1990) and by Goldberg and Sipser (1991) that considered it as a special kind of optimal compression.

The aim of this thesis is to determine classes of NP relations for which random generation and approximate counting problems admit an efficient solution. Since efficient rank implies efficient random generation, we first investigate some classes of NP relations admitting efficient *ranking*. On the other hand, there are situations in which efficient random generation is possible even when ranking is computationally infeasible. We introduce the notion of *ambiguous description* as a tool for random generation and approximate counting in such cases and show, in particular, some applications to the case of formal languages. Finally, we discuss a limit of an heuristic for *combinatorial optimization* problems based on the random initialization of local search algorithms showing that derandomizing such heuristic can be, in some cases, \sharp P-hard. More details follow.

Ranking. We extend some results about ranking for formal languages to the case of NP relations, a fact that allows us to introduce two new classes of relations admitting efficient (i.e., polynomial time) random uniform generation. In particular, we prove that the classes of NP relations accepted by

- (i) unambiguous *auxiliary pushdown automata* working in polynomial time, and

¹by NP relations here we mean subsets $R \subseteq \Sigma^* \times \Sigma^*$ such that, for every $\alpha, \beta \in \Sigma^*$, $|\beta|$ is polynomially related to $|\alpha|$ and the predicate $\alpha R \beta$ can be decided in time polynomial in $|\alpha|$ (where Σ is some finite alphabet). For a more detailed discussion, see Section 1.2.

- (ii) nondeterministic *Turing machines* using $s(n)$ space, $i(n)$ inversions and having ambiguity $d(n)$ with $s(n) \cdot i(n) \cdot d(n) = O(\log n)$,

are such that their rank functions can be computed in polynomial time. These results follow from the techniques used by Huynh (1990) and, respectively, Bertoni, Mereghetti, and Pighizzini (1994) for the ranking of formal languages. Hence, since we have also proved that efficient rank implies efficient random generation, the above classes of relations both admit polynomial time random uniform generation.

Ambiguous descriptions. To deal with the case in which ranking is computationally infeasible, we introduce a simple notion of *description* of a combinatorial structure, together with a corresponding notion of *ambiguity*, and study the problem of uniform random generation and approximate counting for structures endowed with such descriptions. We prove a general result stating that if a structure \mathcal{S} has a description \mathcal{T} with polynomially bounded ambiguity and \mathcal{T} admits a polynomial time *uniform random generator* (u.r.g.), then also \mathcal{S} admits a u.r.g. working in polynomial time. If, moreover, the counting problem for \mathcal{T} is solvable in polynomial time, then \mathcal{S} admits a fully polynomial time *randomized approximation scheme* (r.a.s.) for its counting problem. Here, the proofs are based on the Karp-Luby technique for sampling from a union of sets (Karp, Luby, and Madras, 1989) and on Hoeffding's inequality (Hoeffding, 1963), a classical tool for bounding the tail probability of the sum of independent bounded random variables. Such general results can be applied to various classes of languages:

- (i) We show that, for *finitely ambiguous context-free languages*, a word of length n can be generated uniformly at random in $O(n^2 \log n)$ time and $O(n^2)$ space, using $O(n^2 \log n)$ random bits. We observe that, in our model of computation, the same bounds for time and random bits are obtained for the uniform random generation of unambiguous c.f. language (Goldwurm, 1995). Similar bounds are obtained for the corresponding randomized approximation scheme. To prove these results we show in detail a multiplicity version of Earley's algorithm for context-free recognition Earley (1970). We prove that, for finitely ambiguous c.f. languages, the number of derivation trees of an input word of size n can be computed in $O(n^2 \log n)$ time and $O(n^2)$ space;
- (ii) We show how to generate, uniformly at random, words from languages accepted by *one-way nondeterministic auxiliary push-down automata* working in polynomial time and using a logarithmic amount of work-space (Cook, 1970, 1971; Brandenburg, 1977). Also in this case, we obtain polynomial time u.r.g. and r.a.s. whenever the automaton has a polynomial number of accepting computations for each input word. Notice that such results hold in particular for polynomially ambiguous context-free languages.
- (iii) We consider the uniform random generation and approximate counting of *rational trace languages* (Diekert and Rozenberg, 1995). Finitely ambiguous rational trace languages (Bertoni, Mauri, and Sakarovitch, 1982; Sakarovitch, 1987) admit u.r.g. and r.a.s. of the same time complexity of the algorithms for their recognition problem. Analogously, we obtain polynomial time u.r.g. and r.a.s. for the rational trace languages that are polynomially ambiguous.

Derandomization and combinatorial optimization. We focus our attention on an heuristic suggested by Grossi (1999) for improving *local search* algorithms. The basic idea is to use a two phase (randomized) algorithm which in a first phase generates uniformly at random some feasible solution and then, starting from the best solution so obtained, performs a local search

phase. For a large class of problems, such heuristic is known to give (randomized) approximation algorithm with a constant performance ratio. We study in detail the case in which the objective function of the combinatorial optimization problem is represented by means of arithmetic circuits: this is a quite common situation and applies to a lot of natural problems. Sufficient conditions are known in such a case to prove that the first phase of the heuristic can be “derandomized” thus yielding to a deterministic approximation algorithm. On the other hand, by investigating a problem of transformation between arithmetic circuits, we bring to light a limit of this approach and show that “derandomizing” the first phase of the heuristic is, in some cases, a $\sharp P$ -hard problem.

Structure of the chapters.

This thesis is organized as follows. In Chapter 1 we introduce some preliminary notions and definitions; in particular, in Section 1.1 we present our model of computation: the PrRAM, which is a probabilistic version of the standard random access machine model endowed in addition with an unbiased coin tossing device. In Section 1.2 we recall the definition of p -relation (Jerrum et al., 1986), a fundamental tool to describe formally the problems we deal with in this work. Section 1.3 concludes the chapter with a short survey on the known results about approximate counting and random generation, as discussed by Jerrum et al. (1986); Jerrum and Sinclair (1989).

Chapter 2 shows how the cited results on *ranking* for formal languages can be applied to the random generation problem. First of all, in Section 2.1 the standard notion of ranking is recalled and a proposed extension to the case of relations is given; hence, we prove (Theorem 2.1.1) that if a p -relation admits efficient (i.e., polynomial time) ranking then it also admits efficient random uniform generation. Hence, in Section 2.2 we extend the result of Bertoni et al. (1994) about the ranking of languages accepted by *Turing machines with simultaneous complexity bounds* to p -relations (Theorem 2.2.1); similarly, in Section 2.3 we extend the result of Huynh (1990) about the ranking of languages accepted by unambiguous *one-way auxiliary pushdown automata* working in polynomial time to p -relations (Theorem 2.3.1). Thanks to Theorem 2.1.1, these last results yield to two new classes of p -relations admitting efficient uniform random generation (Corollaries 2.2.1 and 2.3.1).

In Chapter 3, for the sake of simplicity, we turn our attention to combinatorial structures introducing a general paradigm that, under suitable hypotheses, leads to polynomial time algorithms both for random generation and approximate counting problems. We begin by recalling some useful definition in Section 3.1, adapting in particular to the case of combinatorial structures the notions of *uniform random generator* (u.r.g.) and *randomized approximation scheme* (r.a.s.) given by Jerrum et al. (1986) for the case of p -relations. In Section 3.2 we introduce a simple notion of *description* for a combinatorial structure together with a related definition of *ambiguity* and prove the existence of a polynomial time u.r.g. (Theorem 3.2.1) and of a fully polynomial r.a.s. (Theorem 3.2.2) for combinatorial structures admitting suitable (ambiguous) descriptions. To elucidate such results, Section 3.3 presents some simple application of our general paradigm to the case of union and product of combinatorial structures.

Chapter 4 gives less trivial applications of the results of the previous chapter. In Section 4.1 we show a polynomial time u.r.g. and a fully polynomial time r.a.s. for polynomially ambiguous *context-free languages* (Theorem 4.1.2). Notice that even for finitely ambiguous context-

free languages exact counting is known to be $\sharp P_1$ -complete² (Bertoni, Goldwurm, and Sabadini, 1991b)). Our results depend on a multiplicity version of Earley’s algorithm for context-free recognition; in particular, for finitely ambiguous c.f. languages, we give an algorithm computing the number of derivation trees of an input word of size n in $O(n^2 \log n)$ time and $O(n^2)$ space (Proposition 4.1.1). Then, in Section 4.2, we show that the class of languages accepted by polynomially ambiguous *one-way auxiliary pushdown automata* working in polynomial time admits a polynomial time u.r.g. and a fully polynomial time r.a.s. (Theorem 4.2.1). Finally, in Section 4.3, we give a polynomial time u.r.g. and a fully polynomial time r.a.s. for polynomially ambiguous *rational trace languages* (Theorem 4.3.1); in particular, in the case of finitely ambiguous rational trace languages, such algorithms have the same time complexity as the algorithms for their recognition problem.

We came back to the more general setting of p -relations in Chapter 5, where we suggest how the results given for the case of combinatorial structures can be generalized to the case of p -relations. More precisely, in Section 5.1 we propose a generalization of Theorem 2.1.1 to the case of p -relations (Corollary 5.1.1) which, in particular, leads to a stronger version of Theorem 2.3.1: this allows one to define a broader class of p -relations admitting polynomial time uniform random generations in terms of relations accepted by polynomially ambiguous one-way pushdown automata working in polynomial time (Corollary 5.1.2). Sections 5.2 and 5.3 conclude the chapter by discussing some closure properties with respect to the union and complement of the classes of p -relations admitting efficient ranking and/or random generation.

As a very different applications of our results, in Chapter 6, we consider the case of (NP) *combinatorial optimization problems*. After the very short survey of basic notions and definitions of Section 6.1, we focus our attention to the *local search* paradigm in Section 6.2, where we recall the approach of Grossi (1999) for improving local search algorithms. The determination of new classes of p -relations admitting polynomial time uniform random generators allows us to introduce a class of NP optimization problems admitting a polynomial time randomized approximation algorithm with a constant performance ratio (Corollary 6.3.1). On the other hand, in Section 6.4, we show some negative results about “derandomization” and local search. By studying a problem of transformation between arithmetic circuits, we show that, in some cases, derandomizing can be as hard as solving a $\sharp P$ -complete problem (Theorems 6.4.1 and 6.4.2).

²the class $\sharp P_1$ is the restriction of $\sharp P$ to functions having unary inputs.

Per l'amore della bislunga ho tagliato dieci giovani alberi pioppi e glabri. Di tanti che erano ne ho fatto cinque cento fogli di carta bianchi e io su quelli ho scritto giorni e mesi per fare una storia. Ora che voi la leggete, sapete se vale o non vale quei pioppi padani e il tempo, la vita nei mesi, di un uomo.

Maurizio Maggiani, *Màuri màuri*

CONTENTS

1	Preliminary Notions	1
1.1	The Model of Computation	1
1.1.1	The classic RAM model	2
1.1.2	The PrRAM model	5
1.1.3	An example of algorithm	7
1.2	The Notion of p -Relation	9
1.2.1	p -Relations	10
1.2.2	Self-reducible p -relations	10
1.3	A Short Survey	11
1.3.1	Basic definitions	11
1.3.2	A theoretical solution to random generation	13
1.3.3	Relationships with other versions of combinatorial problems	13
1.3.4	Almost uniform generation and approximate counting	14
2	Uniform Random Generation Through Ranking	17
2.1	Ranking p -Relations and Uniform Random Generation	17
2.1.1	The ranking for formal languages	18
2.1.2	The case of relations	18
2.1.3	The unranking	19
2.1.4	A uniform random generator	20
2.2	Turing Machines with Simultaneous Complexity Bounds	20
2.2.1	The case of formal languages	20
2.2.2	An extension to p -relations	21
2.3	One-way Auxiliary Pushdown Automata	24
2.3.1	The case of formal languages	24

2.3.2	An extension to p -relations	25
3	Ambiguous Descriptions	29
3.1	Combinatorial Structures	29
3.1.1	Uniform random generator	30
3.1.2	Randomized approximation scheme	31
3.1.3	Randomized exact counter	32
3.1.4	An example: regular languages	33
3.2	Ambiguous Description	33
3.2.1	Uniform random generation	35
3.2.2	Randomized counting	36
3.3	Simple Applications	39
4	Applications to Formal Languages	41
4.1	Context-Free Languages	41
4.1.1	Uniform random generation of derivation trees	42
4.1.2	Earley's algorithm for counting derivations	44
4.1.3	Inherently ambiguous context-free languages	47
4.1.4	The grammar as a part of the input	48
4.2	One-way Nondeterministic Auxiliary Pushdown Automata	49
4.3	Rational Trace Languages	52
5	Some Remarks	55
5.1	From Combinatorial Structures to p -Relations	55
5.2	The p -Union	56
5.3	The p -Complement	57
6	Applications to Combinatorial Optimization	59
6.1	Some Basic Definitions and Properties	60
6.1.1	The class NPO	60
6.1.2	Approximation algorithms and approximation classes	61
6.1.3	Logically definable approximation classes	62
6.1.4	Approximation preserving reductions	63
6.2	Local Search	64
6.2.1	The basic idea	64
6.2.2	Local search and NPO problems	65

6.3	The “Expectation-Guaranteed” Class	66
6.3.1	Randomized choice of initial solutions	66
6.3.2	Derandomization: the EG class	68
6.3.3	The method of conditional expectation	69
6.3.4	Derandomization and the EG class	71
6.4	Derandomization and Arithmetic Circuits	72
6.4.1	Polynomials and arithmetic circuits	72
6.4.2	Back to our problem	74
6.5	Conclusions	77
7	Conclusions and Open Problems	79
A	Technicalities	81
A.1	Some Lemmas	81
A.2	Probabilistic Detour	82
A.2.1	Hoeffding’s inequality	82
A.2.2	Improving randomized algorithms	84
A.2.3	The method of conditional expectation	88

CHAPTER 1

PRELIMINARY NOTIONS

Bisognava concludere. Manifestai alla contessina Delrio ciò che sentivo di non poterle dissimulare più a lungo. Si rassegnasse all'idea: le diagonali del parallelogrammo si secano nel loro punto mediano. E non è tutto: esse ne dividono l'area in quattro triangoli equivalenti.

Carlo Emilio Gadda, *La Madonna dei filosofi*

In this chapter we introduce some preliminary notion and definition. First of all, we discuss in some detail the model of computation we refer to in the rest of this work. To this aim, we present the RAM model together with a proposed probabilistic extension of such model: the PrRAM; we also give an example of algorithm, discussing its implementation on such model in order to elucidate some of the concepts here introduced.

Then, we recall the definition of p -relation and discuss its relevance as a unifying tool for the formal definition and analysis of various versions of combinatorial problems, in particular with regard to the notion of self-reducibility.

Finally, we present a brief survey of known results on random generation of p -relations, discussing in particular the relationship between (almost) uniform random generation and approximate counting.

1.1 The Model of Computation

One of the fundamental issues arising in the design and analysis of algorithms and in the investigation of the inherent computational difficulty of various problems is the choice of the formal model of computation. Even though it is well known that all the “reasonable” computational models are polynomially related with respect to the time and space complexity, nonetheless each model enjoys some peculiar properties that can help us in the choice.

The Turing machine model, for instance, with its primitive instruction repertoire, is suitable for the investigation of negative results and lower bounds typically arising in the structural complexity investigations. On the other hand, random access machines, or the arithmetic machine model, lead to a more natural and easily understandable notation for the description and design of high level algorithms.

1.1.1 The classic RAM model

Since the main aim of this work is to design efficient algorithms rather than to provide lower bounds, we focus our attention on the RAM model, that we now briefly recall (for a complete reference, and for a discussion on the Turing machine model, see (Aho, Hopcroft, and Ullman, 1974)).

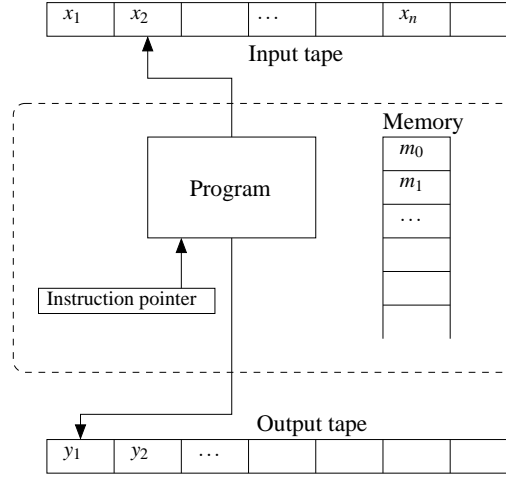


Figure 1.1: A random access machine.

A *random access machine* (RAM) consists of a read-only *input tape*, a write-only *output tape*, a *memory*, and a *program* (Figure 1.1). The input tape is a sequence of cells each of which holds an integer; whenever an integer is read from the input tape, the tape head moves one cell to the right. The output tape is a (potentially infinite) sequence of cells that are initially all blank; when a write instruction is executed, an integer is printed in the cell of the output tape currently under the output tape head, hence the head is moved one cell to the right. The memory consists of a sequence of registers m_0, m_1, \dots each of which can hold an integer; no upper bound is placed on the number of registers that can be used.

The program is a fixed sequence of (optionally) labelled instructions that are not modified during the execution. We assume that there are input-output instructions manipulating the tapes (read and write), instructions moving data across memory (load and store), arithmetic instructions (add, sub, mult, and div), branching instructions (jump, jzero and jgtz) and, finally, halt to stop the computation. Each instruction manipulating data may operate in an *indirect addressing* mode (for example, for indexing arrays); the operand of such instructions can be “ $=i$ ” to indicate the integer $i \in \mathbf{Z}$, “ i ” to indicate the content of register m_i , or “ $*i$ ” to denote the content of m_j where j is the content of m_i .

Semantic

The *semantic* of the computation of a RAM can be easily defined with the help of two elements: an *instruction pointer* that determines the next instruction of the program to be executed and a *memory map* $M : \mathbf{N} \rightarrow \mathbf{Z}$ that gives, for every $i \in \mathbf{N}$, the contents $M(i)$ of the i -th register, that is the integer stored in m_i . At the beginning of the computation, both tape heads scan the respective first (leftmost) cell, the output tape cells are all blank, all the registers are zero (i.e., $M(i) = 0$ for all $i \in \mathbf{N}$) and the instruction pointer points to the first instruction of the program. During the execution the instruction pointer is modified so that the instructions in the program are executed in sequential order unless `jump` is executed, `jzero` is executed when $M(0) = 0$, or `jgtz` is executed when $M(0) \geq 0$: in these cases, the instruction pointer is modified according to the label that follows the jumping instruction.

Instruction	Meaning
<code>read i</code>	$M(i)$ takes as value the integer in the current input cell
<code>read $*i$</code>	$M(M(i))$ takes as value the integer in the current input cell
<code>write a</code>	$V(a)$ is written in the current output cell
<code>load a</code>	$M(0)$ takes value $V(a)$
<code>store i</code>	$M(i)$ takes value $M(0)$
<code>store $*i$</code>	$M(M(i))$ takes value $M(0)$
<code>add a</code>	$M(0)$ takes value $M(0) + V(a)$
<code>sub a</code>	$M(0)$ takes value $M(0) - V(a)$
<code>mult a</code>	$M(0)$ takes value $M(0) \times V(a)$
<code>div a</code>	$M(0)$ takes value $\lfloor M(0)/V(a) \rfloor$
<code>jump b</code>	the instruction pointer points to b
<code>jzero b</code>	if $M(0) = 0$, then the instruction pointer points to b , else it points to the next instruction in the program
<code>jgtz b</code>	if $M(0) \geq 0$, then the instruction pointer points to b , else it points to the next instruction in the program
<code>halt</code>	the execution stops

Table 1.1: Meaning of RAM instructions.

The meaning of the various instructions should be clear to anyone who is familiar with some kind of assembly language and is briefly recalled in Table 1.1 where, for the sake of brevity, we have denoted by $V(a)$ the *value* of operand a defined as follows: $V(=i) = i$, $V(i) = M(i)$ and $V(*i) = M(M(i))$.

Given the semantic described so far, the computation of a RAM can be essentially seen as a (partial) function from the content of its input tape to the (non-blank) cells of its output tape (defined whenever the computation halts). More formally, given a RAM P , with an abuse of notation, we denote by P also the (partial) function $P : \mathbf{N}^* \rightarrow \mathbf{N}^*$ such that $P(x_1, \dots, x_n)$ is the content of the output tape of P (omitting the trailing blank cells) whenever the computation of P starting with $x_1 \dots x_n$ on the input tape halts; otherwise $P(x_1, \dots, x_n)$ is undefined. The class of functions so defined can be related to the class of *partial recursive functions* (Davis, 1958; Rogers, 1967), it is in fact possible to show that (Aho et al., 1974)

Proposition 1.1.1 *The class of (partial) functions that can be computed in the RAM model*

coincides with the class of recursive (partial) functions.

Computational complexity

To define the *computational complexity* in the RAM model we need to specify the time required to execute each instruction and the space used by each register; to this end, two choices are usually made up in the literature: the *uniform cost criterion* and the *logarithmic cost criterion* (Aho et al., 1974). We briefly recall the latter since we shall use it for the analysis of the algorithms that will be presented in this work. Let $l(i)$ be the logarithmic function on the integers defined by¹

$$l(i) = \begin{cases} \lfloor \log(i) \rfloor + 1 & \text{if } i > 0, \\ 1 & \text{if } i = 0 \end{cases}$$

and define the (logarithmic) cost $t(a)$ of accessing the operand a as $t(=i) = l(i)$, $t(i) = l(i) + l(M(i))$ and $t(*i) = l(i) + l(M(i)) + l(M(M(i)))$. Then the cost of the RAM instruction is given in Table 1.2.

Instruction	Cost
read i	$l(\text{input}) + l(i)$
read $*i$	$l(\text{input}) + l(i) + l(M(i))$
write a	$t(a)$
load a	$t(a)$
store i	$l(M(0)) + l(i)$
store $*i$	$l(M(0)) + l(i) + l(M(i))$
add a	$l(M(0)) + t(a)$
sub a	$l(M(0)) + t(a)$
mult a	$l(M(0)) + t(a)$
div a	$l(M(0)) + t(a)$
jump b	1
jzero b	$l(M(0))$
jgtz b	$l(M(0))$
halt	1

Table 1.2: Logarithmic cost of RAM instructions.

Some remarks

We conclude this section observing that the choice of the RAM model under logarithmic cost criterion enjoys two important properties we look for in the design and analysis of the algorithms presented in this work. First of all, the high-level language and structure of the RAM avoids the need of the boring specifications usually arising when using the Turing machine model and moreover it allows us to design algorithms that are easier to understand and more similar to their implementation in today's programming languages. On the other and, by taking into reasonable account the size of the operands, the logarithmic cost criterion will allow the analysis of our

¹ All the logarithms in this thesis are in base 2, unless otherwise specified.

algorithms to be realistic in the precise sense of the following proposition. Recall that two functions f, g are said to be *polynomially related* if two polynomials p, q exist such that $f(n) \leq p(g(n))$ and $g(n) \leq q(f(n))$ for every $n > 0$. Then, it is well known that (Aho et al., 1974)

Proposition 1.1.2 *The time (and space) cost in the RAM model under logarithmic cost criterion and in the (multi-tape) Turing machine model are (respectively) polynomially related.*

This means that every algorithm we design on a RAM can be effectively implemented by some fixed hardware so that the logarithmic time required by the algorithm on the RAM is polynomially related to the time required by the hardware to actually carry on the computation.

1.1.2 The PrRAM model

Since a large part of this work is devoted to randomized algorithms, we need now to introduce some probabilistic model of computation. Concerning the source of randomness, two choices are usually present in the literature. The first, which characterizes the probabilistic Turing machine model (Gill, 1977), is to enrich the deterministic model of computation with a device able to toss an unbiased coin, that is to allow branches during the computation so that each of the two branch can be taken with equal probability. The second choice, usually adopted when considering models of computation similar to the RAM, is to enrich the deterministic model with one (or more) coin of (different) rational, or even arbitrary, bias. It is clear that the two models have very different computational power: if one has only a fair coin, then even the simple task of choosing with equal probability between three numbers, that is rolling a 3-sided die, has no algorithm that always terminates!

This asymmetry leads to different approaches in the design of probabilistic algorithms. The first possibility is to focus on algorithms working in *bounded time*, which always terminate but that, specially when using the unbiased coin model, can fail to produce their output exactly according to a desired distribution. If this happens, there are two possibilities: a special symbol can be output denoting the failure of the algorithm, or the algorithm may simply try to approximate the desired distribution up to some specified precision degree. On the other hand, if one focuses on *bounded expected time* algorithms, then even in the unbiased coin model one can look for (possibly very slow, or even nonterminating) algorithms whose output exactly follows a desired distribution, or can choose to trade off the computation time with the approximation precision. Some examples of the bounded time approach can be found for instance in the works of Jerrum et al. (1986); Jerrum and Sinclair (1989) and of Feldman, Impagliazzo, Naor, Nisan, Rudich, and Shamir (1993), while examples of the expected time approach can be found for instance in the works of Alonso (1994); Barucci, Pinzani, and Sprugnoli (1992).

Without going further into details, one has to notice that a bounded time algorithm using a special symbol to denote its failure can, in principle, be iterated until success, leading to a bounded expected time algorithm; on the other hand, a bounded expected time algorithm whose output eventually follows exactly a desired distribution can, in principle, be prematurely halted leading to a bounded time approximating algorithm. As one can conclude even from this very short presentation, the difference between these two approaches is very subtle and often the choice between them is a matter of taste, or opportunity, unless feasibility of the computational model is considered.

As suggested by Jerrum et al. (1986), allowing the tossing of an arbitrary (rationally) biased coin whose bias depends on the input violates the philosophy of Turing machines, because the

coin tossing steps of different biases cannot be implemented with a fixed amount of hardware and in a fixed time. Feldman et al. (1993) in fact show that at least two coins of rational bias are necessary to simulate a n -sided die, unless n is a power of two, and that the simulation of every coin of rational bias whose denominator is less than n with d coin flips requires at least $\Omega(\log n / \log d)$ differently biased rational coins.

Thus, to follow the principle of a feasible model of computation we have stated in the previous section, enforced by choosing the logarithmic cost criterion to allow a realistic analysis of our algorithms, our choice of the source of randomness will be in favor of the unbiased coin model.

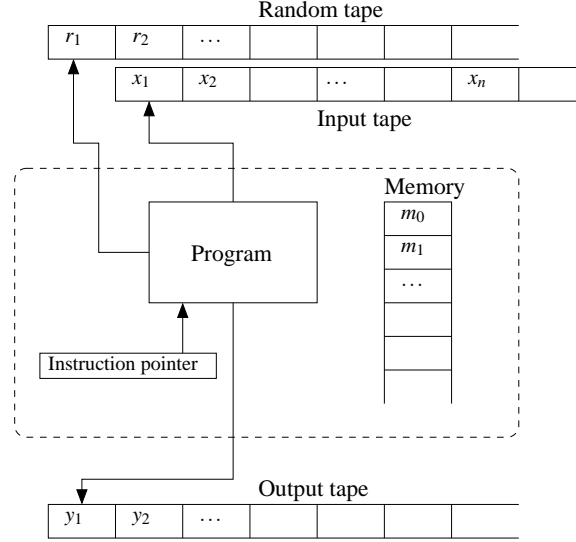


Figure 1.2: A PrRAM.

Our model of *probabilistic random access machine* (PrRAM) consists of a RAM machine endowed in addition with a read-only *random tape* (Figure 1.2) and a further instruction `rnd` to operate on it. The random tape is a sequence of cells each containing 0 or 1; when the instruction “`rnd a`” is executed, the random tape head scans $V(a)$ cell and $M(0)$ takes as value the integer number given, in binary notation, by the juxtaposition of the read cells. More formally, if $R(j)$ denotes the content of the j -th random tape cell r_j , when the random tape head is positioned on r_i and “`rnd a`” is executed, then $M(0) = \sum_{j=0}^{V(a)-1} 2^j R(i+j)$ and the random tape head is positioned on $r_{i+V(a)}$.

Instruction	Meaning
<code>rnd a</code>	$M(0)$ takes value $\sum_{j=0}^{V(a)-1} 2^j R(i+j)$.

Table 1.3: Meaning of PrRAM instructions.

Semantic

To define the *semantic* of a PrRAM computation, we introduce the following notation: let $P(x_1, \dots, x_n; r_1, r_2, \dots)$ be the (partial) function denoting the contents of the output tape (omitting the trailing blank cells) at the end of the computation of the PrRAM P having $x_1 \dots x_n$ on the input tape and $r_1 r_2 \dots$ on the random tape (whenever the computation halts, otherwise we let $P(x_1, \dots, x_n; r_1, r_2, \dots)$ undefined). Since r_1, r_2, \dots are fixed, the computation can be considered deterministic and its semantic can be easily derived by the semantic of the RAM model discussed above and from the given meaning of the `rnd` instruction. Then $P(x_1, \dots, x_n)$ is defined as the random variable $P(x_1, \dots, x_n; R_1, R_2, \dots)$ where R_1, R_2, \dots are independent and identically distributed Bernoulli random variables such that $\Pr\{R_i = 1\} = \Pr\{R_i = 0\} = 1/2$ for $i \in \mathbb{N}$.

Computational complexity

Again, to define the computational complexity in the PrRAM model we need to specify the time required to execute the `rnd` instruction; to take into account the dimension of the operand also in this case, we also assume that the cost of a “`rnd a`” instruction is $t(a)$ (where t is defined as in the previous section). Moreover, together with the usual time and space complexity measures, for the PrRAM model we can also consider explicitly the number of cells scanned on the random tape, a quantity which we call the *number of random bits* used by the computation.

Instruction	Cost
<code>rnd a</code>	$t(a)$

Table 1.4: Cost of PrRAM instructions.

Some remarks

We want to highlight again that, thanks to the choice for the source of randomness adopted in the PrRAM model discussed in this section, we believe to have attained our purpose to suggest a model of computation at the same time feasible and of sufficient high-level to describe in a natural fashion the algorithms we will design in this work. As for the RAM case, it is straightforward to check that the following holds

Proposition 1.1.3 *The time (and space) cost PrRAM model (under logarithmic cost criterion) and the (multi-tape) probabilistic Turing machine model are polynomially related.*

1.1.3 An example of algorithm

All the algorithms presented in the following will be described using a high-level language, usually known as “Pidgin ALGOL” (for its informal description, see (Aho et al., 1974)); what is essential here, is that such algorithms can be translated into a RAM program in a straightforward manner, hence we shall never be concerned with the details of this “compilation” process. In our Pidgin ALGOL we allow the use of every kind of usual mathematical statement and programming language construct such as expressions, conditions, statements and (recursive) procedures;

moreover, the language has no fixed set of data types: variables can represent integers, strings and arrays, or even more complex objects such as sets, lists and graphs. No attempt will be made here to give a precise definition of all the constructs used as long as their meaning will be clear and their translation into PrRAM code is evident from the context.

Uniform random generation of integers

As an example of what stated in this section, we give an algorithm (Algorithm 1.1) for the uniform random generation of an integer in the range $1, \dots, N$. We discuss its implementation on a PrRAM and the consequent analysis of its complexity and correctness. We need to fix some $0 < \delta < 1$ as an upper bound to the probability that the algorithm fails to give output according to the uniform distribution, a fact that the algorithm will signal by use of the special symbol \perp (we recall that this depends on our choice of an unbiased coin as a source of randomness).

```

input  $N$ 
 $r \leftarrow \perp, i \leftarrow 0$ 
while  $i < \lceil \log(1/\delta) \rceil$  and  $r = \perp$  do
     $i \leftarrow i + 1$ 
    generate  $u \in \{1, \dots, 2^{\lceil \log N \rceil}\}$  uniformly at random
    if  $u \leq N$  then  $r \leftarrow u$ 
output  $r$ .

```

Algorithm 1.1: a uniform random integer numbers generator.

Consider first of all the implementation. The “deterministic” statements are all very simple to implement on a RAM, as it is easy to verify; in particular, $\lceil \log \delta \rceil$ is a constant (independent of the input), while $\lceil \log N \rceil$ can be computed in $O(\log N \log \log N)$ time (see Lemma A.1.3 for details).

On the other hand, for every $m \in \mathbb{N}$, the “probabilistic” statement “**generate** $u \in \{1, \dots, 2^m\}$ *uniformly at random*”, can be implemented on a PrRAM essentially using just a “**rnd** m ” instruction.

Concerning the correctness of the algorithm, if we denote by $R(N)$ the random variable giving the output of the PrRAM R implementing the algorithm, we are able to prove that, for every $N \in \mathbb{N}$,

$$\Pr\{R(N) = \perp\} < \delta, \text{ and} \quad (1.1)$$

$$\Pr\{R(N) = n \mid R(N) \neq \perp\} = 1/N, \text{ for every } n \in \{1, \dots, N\}. \quad (1.2)$$

Here, equation (1.1) substantially tells that the algorithm gives a “correct” output with probability $1 - \delta$, and equation (1.2) states that, if the algorithm terminates “correctly”, then its output is actually uniformly distributed over the desired range. The first equation follows by noting that the output of the algorithm is \perp iff u is greater than N for $\lceil \log(1/\delta) \rceil$ times:

$$\begin{aligned}
 \Pr\{R(N) = \perp\} &= \Pr\{u > N\}^{\lceil \log(1/\delta) \rceil} \\
 &= \left(1 - \frac{N}{2^{\lceil \log N \rceil}}\right)^{\lceil \log(1/\delta) \rceil} \\
 &< 1/2^{\lceil \log(1/\delta) \rceil} \leq \delta,
 \end{aligned}$$

where the first inequality follows from the fact that $N/2^{\lceil \log N \rceil} > 1/2$, for every $N > 0$. Moreover, for every $n \in \{1, \dots, N\}$,

$$\Pr\{u = n \mid u \leq N\} = \frac{\Pr\{u = n\}}{\Pr\{u \leq N\}} = \frac{1}{2^{\lceil \log N \rceil}} \left(\frac{N}{2^{\lceil \log N \rceil}} \right)^{-1} = \frac{1}{N}.$$

Since the output of the algorithm, whenever it is different from \perp , is distributed as u conditionally to the fact that $u \leq N$, by the previous equality, we can finally derive equation (1.1). We can summarize the result of this section as

Proposition 1.1.4 *For every $0 < \delta < 1$, there exists a PrRAM which, on input $N \in \mathbf{N}$, gives in output an integer chosen uniformly at random in $\{1, \dots, N\}$ with probability $1 - \delta$, taking $O(\log N(\log \log N - \log \delta))$ time.*

1.2 The Notion of p -Relation

We now recall a notion that allows us to formalize several kind of combinatorial problems. The use of *languages*, i.e., subsets of the free monoid Σ^* (where Σ is some finite alphabet), to describe formally combinatorial problems is well established (see, for instance, Garey and Johnson, 1979). In a similar fashion, one can also use *relations* in $\Sigma^* \times \Sigma^*$ to describe, with “more richness”, such problems. As an example, for a suitable encoding, consider the relations

- (1) (F, τ) , where F is a boolean formula and τ is one of its satisfying truth assignments;
- (2) (G, K) , where G is a graph and K is one of its cliques²;
- (3) (n, P) , where $n \in \mathbf{N}$ is a positive integer and P is the set of partitions of n ;
- (4) (G, K) , where K is a (induced) subgraph³ of G .

The use of relations instead of languages is more rich in the sense that they enable us to define several “versions” of a combinatorial problem in a natural way. Given some $R \subseteq \Sigma^* \times \Sigma^*$, define $R(\alpha) = \{\beta \in \Sigma^* : \alpha R \beta\}$, for every $\alpha \in \Sigma^*$. Then, for some fixed relation R , one have the following kinds of problems

- (a) *decision*: given $\alpha \in \Sigma^*$, decide whether there exists some $\beta \in R(\alpha)$;
- (b) *construction*: given $\alpha \in \Sigma^*$, construct, if it exists, some $\beta \in R(\alpha)$;
- (c) *counting*: given $\alpha \in \Sigma^*$, count the number of $\beta \in R(\alpha)$;
- (d) *random generation*: given $\alpha \in \Sigma^*$, generate an element of $R(\alpha)$ uniformly at random.

Note on passing that, as we will see in Chapter 6, also combinatorial *optimization* problems can be defined in a natural way using relations.

Such a unifying view of combinatorial problems by means of relations has appeared in the literature in the form of *string relations* by Garey and Johnson (1979) and of *search functions* by Valiant (1978). It is used in this thesis mainly to relate generation problems, which are the main subject of this work, to more familiar combinatorial problems such as existence and counting.

² K is a subset of the vertices of G such that every pair of vertices in K are joined by an edge in G .

³ K is an induced subgraph of G if it is isomorphic to a (proper) subgraph of G .

1.2.1 *p*-Relations

A class of relations in $\Sigma^* \times \Sigma^*$ of particular interest is the subclass of the relations that can be “checked efficiently”; more formally, Jerrum et al. (1986) give the following:

Definition 1.2.1 *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a p -relation if*

- (i) *there exists a polynomial p for which $|\beta| = p(|\alpha|)$ for every $\alpha, \beta \in \Sigma^*$ such that $\alpha R \beta$;*
- (ii) *the relation R can be decided⁴ in polynomial time.*

Two remarks are in order. First, we observe that, without loss of generality, for the sake of simplicity we fix $|\beta| = p(|\alpha|)$ while the original definition of Jerrum et al. (1986) asks only that $|\beta| \leq p(|\alpha|)$. Second, here and in the following, by an abuse of notation, for every p -relation, by the same letter “ p ” we will always (implicitly) denote the polynomial whose existence is required by the definition.

Note, for instance, that examples (1) and (2) above are obviously p -relations, whereas example (3) evidently violates condition (i) of Definition 1.2.1 and example (4) violates condition (ii) of the same definition unless $P = NP$.

Finally, observe that it is not difficult to realize that the above mentioned versions (a) and (c) of combinatorial problems give rise, in the case of p -relations, to the well known classes NP, of decision problems (Garey and Johnson, 1979) and, respectively, $\#P$, of counting problems (Valiant, 1979).

1.2.2 Self-reducible p -relations

We now recall the definition of *self-reducible* p -relation as introduced by Schnorr (1976); Jerrum et al. (1986):

Definition 1.2.2 *A p -relation $R \subseteq \Sigma^* \times \Sigma^*$ is self-reducible if there exist two polynomial time computable functions $\psi : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ and $\sigma : \Sigma^* \rightarrow \mathbb{N}$ such that, for every $\alpha, \omega, \beta = b_1 \dots b_n \in \Sigma^*$,*

- (i) $\sigma(\alpha) = O(\log |\alpha|)$,
- (ii) $R(\alpha) \neq \emptyset$ implies $\sigma(\alpha) > 0$,
- (iii) $|\psi(\alpha, \omega)| \leq |\alpha|$ and
- (iv) $\alpha R \beta$ iff $\psi(\alpha, b_1 \dots b_{\sigma(\alpha)}) R b_{\sigma(\alpha)+1} \dots b_n$.

Intuitively, a relation is self-reducible if, for every element α of its domain, given a small (logarithmic) prefix $b_1 \dots b_{\sigma(\alpha)}$ of a related β , it is possible to efficiently compute a smaller (w.r.t. α) element of its domain $\psi(\alpha, b_1 \dots b_{\sigma(\alpha)})$ that is in relation with the suffix $b_{\sigma(\alpha)+1} \dots b_n$ of β .

Consider, for instance, the relation of example (1) above. Let F be a boolean formula on the variables v_1, \dots, v_n and $\tau : \{v_1, \dots, v_n\} \rightarrow \{\text{true}, \text{false}\}$ be a satisfying truth assignment for it. By substituting in F the values $\tau(v_1), \dots, \tau(v_{\log n})$ in place of the variables $v_1, \dots, v_{\log n}$, one

⁴that is, the language $\{\alpha \diamond \beta : \alpha, \beta \in \Sigma^*, \alpha R \beta\}$, where \diamond is some symbol not belonging to Σ , belongs to P.

(efficiently) obtains a new formula F' smaller (with less variables, and hence shorter) than F . It is also evident that if $v'_1, \dots, v'_{n-\log n}$ are the variables of F' , then $\tau' : \{v'_1, \dots, v'_{n-\log n}\} \rightarrow \{\text{true}, \text{false}\}$ defined as $\tau'(v'_i) = \tau(v_{i+\log n})$ for $1 \leq i \leq n - \log n$ is a satisfying truth assignment for F' . Hence, such a p -relation is self-reducible.

Not every p -relation is self-reducible

The notion of self-reducibility is very natural and applies to many of the well known combinatorial problems to the extent that Jerrum et al. (1986) state that “problems which cannot be formulated in a self-reducible way seem to be the exception rather than the rule”. Nonetheless, here we want to note that there are some very natural cases of relations which do not enjoy this property and this fact is very relevant for this thesis, as we discuss in the following.

A natural class of p -relations can be defined by means of formal languages. For a fixed $L \subseteq \Sigma^*$, we define the corresponding *slice relation* R_L as $a^n R_L \alpha$ iff $\alpha \in \Sigma^n \cap L$ (for some fixed $a \in \Sigma$).

Take now as example the regular language cd^* of words beginning with a letter c followed by any number of d 's. It is very simple to conclude that its corresponding slice relation is not self-reducible. For every function σ, ψ (since for every $n > 0$ it must be $\sigma(a^n) > 0$), every suffix $b_{\sigma(a^n)+1} \dots b_n$ of cd^* will be made only of d 's, but no word of this kind belongs to the original regular language. Hence, whichever is the length m such that $1^m = \psi(1^n, b_1 \dots b_{\sigma(a^n)})$, condition (iv) of Definition 1.2.2 can never hold.

1.3 A Short Survey

In this section we want to briefly summarize some relationship between uniform random generation, approximate counting and other “versions” of combinatorial problems as discussed by Jerrum et al. (1986); Jerrum and Sinclair (1989).

1.3.1 Basic definitions

First of all, we recall some basic definitions of Jerrum et al. (1986) we here adapt to our notation, for the sake of consistency with the following chapters⁵. We begin with random generation:

Definition 1.3.1 *An algorithm A is an almost uniform generator for a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ iff, for every tolerance $\varepsilon \in (0, 1)$ and $\alpha \in \Sigma^*$ such that $R(\alpha) \neq \emptyset$,*

- (i) *A on input α, ε gives output $A(\alpha, \varepsilon) \in R(\alpha) \cup \{\perp\}$,*
- (ii) *for every⁶ $\beta \in R(\alpha)$, $(1 - \varepsilon)/\#R(\alpha) \leq \Pr\{A(\alpha, \varepsilon) = \beta \mid A(\alpha, \varepsilon) \neq \perp\} \leq (1 + \varepsilon)/\#R(\alpha)$,*
- (iii) *$\Pr\{A(\alpha, \varepsilon) = \perp\} < 1/4$.*

⁵however, it remains an easy task to verify that the all the definitions we give here, are, to our aim, equivalent to the original one.

⁶in the following, to avoid confusion with $|\cdot|$ which will be used to denote both the size of an object in a combinatorial structure and the length of a word in a formal language, we denote by $\#A$ the cardinality of the set A .

Moreover, an almost uniform generator is said fully polynomial iff it works in time polynomial in $|\alpha|$ and $\log(1/\epsilon)$.

Observe that the inclusion of the logarithm means that a fully polynomial almost uniform generator can, at a modest computational expense, achieve an output distribution which is very close to uniform. If we let the tolerance $\epsilon = 0$, we get the following

Definition 1.3.2 An algorithm A is an exact uniform generator for a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ iff, for every $\alpha \in \Sigma^*$ such that $R(\alpha) \neq \emptyset$,

- (i) A on input α gives output $A(\alpha) \in R(\alpha) \cup \{\perp\}$,
- (ii) $\Pr\{A(\alpha) = \beta \mid A(\alpha) \neq \perp\} = 1/\#R(\alpha)$, for every $\beta \in R(\alpha)$, and
- (iii) $\Pr\{A(\alpha, \epsilon) = \perp\} < 1/4$.

We now turn to counting, more precisely, to approximate counting algorithm, in the sense of Stockmeyer (1983); Jerrum et al. (1986).

Definition 1.3.3 An algorithm A is a randomized approximate counter within ratio $\rho : \mathbf{N} \rightarrow \mathbf{R}^+$ for a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ iff, for every $\alpha \in \Sigma^*$ such that $R(\alpha) \neq \emptyset$,

- (i) A on input α gives output $A(\alpha) \in \mathbf{Q} \cup \{\perp\}$,
- (ii) $\Pr\{1/\rho(|\alpha|) \leq A(\alpha)/\#R(\alpha) \leq \rho(|\alpha|) \mid A(\alpha) \neq \perp\} > 3/4$ and
- (iii) $\Pr\{A(\alpha) = \perp\} < 1/4$.

Definition 1.3.4 An algorithm A is a randomized approximation scheme for a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ iff, for every $\alpha \in \Sigma^*$ such that $R(\alpha) \neq \emptyset$ and every $\epsilon \in (0, 1)$,

- (i) A on input α, ϵ gives output $A(\alpha, \epsilon) \in \mathbf{Q} \cup \{\perp\}$,
- (ii) $\Pr\{(1 - \epsilon)\#R(\alpha) \leq A(\alpha, \epsilon) \leq (1 + \epsilon)\#R(\alpha) \mid A(\alpha, \epsilon) \neq \perp\} > 3/4$ and
- (iii) $\Pr\{A(\alpha, \epsilon) = \perp\} < 1/4$.

Moreover, a randomized approximation scheme is said to be fully polynomial time whenever it works in time polynomial in n and $1/\epsilon$.

We conclude this section noting that the constants $1/4$ and $3/4$ present in the definitions of (almost) uniform generator, randomized approximate counter and approximation scheme can be replaced by other suitably chosen constants, leaving such definitions substantially unchanged by essentially the same arguments discussed in Section 3.1.

1.3.2 A theoretical solution to random generation

A widely discussed topic in the literature is the strict relationship existing between the problem of counting and generating combinatorial objects. This relationship is made precise in the work of Jerrum et al. (1986) by means of oracle (probabilistic) Turing machines with oracles in the polynomial hierarchy (Stockmeyer, 1977).

Here, we want only to briefly recall that a *Turing machine with oracle* $O \subseteq \Sigma^*$ is a Turing machine⁷ endowed in addition with an *oracle tape* and a distinguished *query state*. The computation of such machine can be described as for the standard Turing machine except for the fact that when the machine enters the query state, with the string ω written on the oracle tape, the “oracle” answers the question “ $\omega \in O$ ” by replacing, in one single move, the contents of the oracle tape with (a suitable encoding of) “yes/no”. The *polynomial hierarchy* of Stockmeyer is defined building a hierarchy of classes of languages by means of such oracle Turing machines (see (Stockmeyer, 1977) for more details and for a definition of the classes of formal languages Σ_n^P).

For the case of exact uniform generation, Jerrum et al. (1986) prove the following

Proposition 1.3.1 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a p -relation. Then there exists a polynomial time oracle (probabilistic) Turing machine that is an exact uniform generators for R either taking a suitable oracle in $\#P$, or in Σ_2^P .*

In the case of almost uniform generation, a weaker request for the oracle is possible; again Jerrum et al. (1986) show the following

Proposition 1.3.2 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a p -relation. Then there exists an oracle (probabilistic) Turing machine, with a suitable oracle in NP (Σ_1^P), which is an almost uniform generator for R working in time $|\alpha|$ and $1/\epsilon$.*

Observe that a usual computational complexity conjecture is that the classes NP , $\#P$ and Σ_2^P contain *highly intractable* problems (see, for instance Balcázar, Díaz, and Gabarró, 1995); for this reason, the above mentioned results are to be considered mainly of theoretical interest.

1.3.3 Relationships with other versions of combinatorial problems

In this subsection we discuss the relationships of the random generation problem with other versions of combinatorial problems: namely the decision and counting versions.

It is clear that a procedure for counting the elements of $R(\alpha)$ must in particular decide if $R(\alpha) = \emptyset$, and hence counting is computationally at least as hard as decision. The first evidence that the counting version can be harder than the decision one for significant natural problems was proven by Valiant (1979). Consider the p -relation (G, M) where $G = \langle V, E \rangle$ is a graph and $M \subseteq E$ is a *perfect matching*⁸ of G ; Valiant showed that counting the number of perfect matching in a bipartite graph (or, equivalently, to compute the permanent of a $\{0, 1\}$ valued matrix) is $\#P$ -complete and hence likely to be computationally intractable, whereas deciding whether a

⁷see Section 2.2 for further details on Turing machines, and Garey and Johnson (1979) for a formal definition of the oracle model.

⁸a perfect matching of a graph is a set of edges such that every node of the graph is the endpoint of precisely one edge in the match.

bipartite graph contains a perfect matching (or, equivalently, deciding if the permanent of a $\{0, 1\}$ valued matrix is non-zero) is in P, by virtue of the classical “augmenting path” algorithm.

Now we give some evidence that uniform generation is, from the computational complexity point of view, somewhere in between decision and counting.

As an example of the fact that the uniform generation version of a problem can be harder than the construction (and hence decision) one, consider the p -relation $R_1 = (G, c)$ where $G = \langle V, E \rangle$ is a directed graph and c is a directed simple cycle of G . As one can verify, the decision version of the combinatorial problem given by the relation R_1 is easily solvable; even the construction version: given a graph G , to output one of its (directed, simple) cycles, is clearly in P. Nonetheless, Jerrum et al. (1986) proved the following

Proposition 1.3.3 *If there exists a polynomial time uniform generator for R_1 , then $\text{NP} = \text{RP}$.*

Hence, as an example of the fact that the uniform generation version can be easier than the counting one, consider the p -relation $R_2 = (F, \tau)$ where F is a boolean formula in disjunctive normal form (DNF), and τ is one of its satisfying assignments. One can verify (Jerrum et al., 1986) that the counting problem for R_2 is $\#P$ -complete, a fact that comes from a reduction from the analogous problem for conjunctive normal form (CNF) boolean formulas which has been proven $\#P$ -complete by Simon (1977). Nonetheless, Jerrum et al. (1986) prove the following

Proposition 1.3.4 *There exists a polynomial time uniform generator for R_2 .*

1.3.4 Almost uniform generation and approximate counting

Going back to almost uniform generation, one finds again a strong relationship with approximate counting. Let $R \subseteq \Sigma^* \times \Sigma^*$ be a self-reducible p -relation and let ψ and σ be the functions cited in Definition 1.2.2. For some $\alpha \in \Sigma^*$ belonging to the domain of R , define the *tree of derivations* (for more details, see Jerrum and Sinclair, 1989) $T_R(\alpha)$ of R as the tree whose vertices are pairs $v = (\delta, \gamma) \in \Sigma^* \times \Sigma^*$ such that

- (a) the root is (α, ε) , where ε is the empty word;
- (b) for any other node $v = (\delta, \gamma)$
 - if $|\delta| = 1$, then v is a leaf, otherwise
 - if $|\delta| > 1$, then v has a child $v_\omega = (\psi(\delta, \omega), \gamma \cdot \omega)$ for every $\omega \in \Sigma^{\sigma(\gamma)}$ such that $R(\psi(\delta, \omega)) \neq \emptyset$.

It should be clear that the second part of all the vertices are distinct and that the second part of the leaves are precisely the elements of $R(\alpha)$, without repetitions. The bounds on ψ and σ in the definition of self-reducibility ensure that the depth of $T_R(\alpha)$ and the number of children of every vertex of $T_R(\alpha)$ are bounded by a polynomial in $|\alpha|$.

Most known uniform generation algorithms for combinatorial structures (see, for example the book of Nijenhuis and Wilf (1978), or even Algorithm 4.1 of Section 4.1.1) may be viewed as instances of the following reduction to the corresponding counting problem. Given that the structures are described by a self-reducible relation R , select a random path from the root of T_R to a leaf, at each stage choosing the next edge with probability proportional to the number of solutions in the maximal subtree rooted at its lower end. This information may be obtained

from a counter which evaluates the function $\#R(\omega)$ for appropriate ω in the tree. Moreover, by appending a correction process (based on the a posteriori probability of the path), such a procedure can be made to work even if the counter is slightly inaccurate.

More formally, let $R \subseteq \Sigma^* \times \Sigma^*$ be a self-reducible p -relation such that $|\beta| = O(|\alpha|^{\kappa_R})$ if $\alpha R \beta$, for some $\kappa_R > 0$. Then, Jerrum et al. (1986) have proven that

- (i) if there exists a polynomial time randomized approximate counter for R within ratio $1 + |\alpha|^{\kappa_R}$, then there exists a fully polynomial time almost uniform generator for R ;
- (ii) if there exists a polynomial time almost uniform generator for R with tolerance $|\alpha|^{2\kappa_R}$, then there exists a fully polynomial time randomized approximate counter for R .

This, in particular, implies that the following holds

Proposition 1.3.5 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a self-reducible p -relation. Then there exists a fully polynomial time randomized approximate counter for R iff there exists a fully polynomial time almost uniform generator for R .*

On the other hand, when a rather cruder counting information is available (to within a constant factor, say) the above mentioned scheme of random generation through (approximate) counting breaks down owing to the accumulation of errors which are too large to be corrected. In such a case a more flexible and self-correcting approach is taken by Jerrum and Sinclair (1989) in which a random process moves dynamically around the tree T_R , with backtracking allowed. The generator hence views the vertices of the tree as the states of a Markov chain in which there is a non-zero transition probability between two states iff they are adjacent in T_R . The transition probability themselves are computed with the aid of the crude approximate counter. Clearly, all states communicate, so that, leaving aside questions of periodicity, if the chain is allowed to evolve for t steps (from any initial state), the distribution of its final state approaches a unique stationary distribution as $t \rightarrow \infty$. If the transition probabilities are such that the stationary distribution is uniform over the leaves of T_R , then one gets an almost uniform generator by simulating the Markov chain for sufficiently many steps.

The study of the *rapid mixing* property of such a chain, i.e., the speed of the convergence of the chain to the stationary distribution, is one of the main result of Jerrum and Sinclair (1989). Thanks to a very sophisticated analysis, the authors are able to prove that, if $R \subseteq \Sigma^* \times \Sigma^*$ is a self-reducible p -relation for which there exists a polynomial randomized approximate counter within ratio $1 + O(|\alpha|^\kappa)$ for some *arbitrary* $\kappa \in \mathbf{R}$, then

- (i) there exists a fully polynomial almost uniform generator for R , and
- (ii) there exists a fully polynomial approximation scheme for R .

This fact, in particular, proves that approximation counting problems, in the case of p -relation, cannot give rise to an “approximation hierarchy” of relations (as, for instance, in the case of combinatorial optimization problems, as discussed in Section 6.1.2). From the previous results, in fact, one can obtain the following surprising

Proposition 1.3.6 *Let $R \subseteq \Sigma^* \times \Sigma^*$ be a self-reducible p -relation. Let $A \subset \mathbf{R}$ such that there exists a polynomial randomized approximate counter within ratio $1 + O(|\alpha|^\kappa)$ for some $\kappa \in A$. Then, either $A = \emptyset$, or $A = \mathbf{R}$.*

CHAPTER 2

UNIFORM RANDOM GENERATION THROUGH RANKING

Così si potesse dimezzare ogni cosa intera, . . . così ognuno potesse uscire dalla sua ottusa e ignorante interezza. . . Se mai diventerai metà di te stesso, e te l'auguro, ragazzo, capirai cose al di là della comune intelligenza dei cervelli interi. Avrai perso metà di te e del mondo, ma la metà rimasta sarà mille volte più profonda e preziosa.

Italo Calvino, *Il visconte dimezzato*

In this chapter we discuss how to apply the concept of *ranking* to the problem of uniform random generation of p -relations. To this end, we briefly recall the concept of ranking for formal languages and suggest an extension of such concept to the case of relations. Then we prove a theorem stating that if a p -relation is rankable in polynomial time, then it admits a polynomial time uniform random generator. Finally, we extend some known results on ranking for formal languages to the case of p -relations, a fact that allows us to introduce two new classes of polynomial time rankable p -relations.

2.1 Ranking p -Relations and Uniform Random Generation

Suppose that, given a totally ordered set of objects, we are able to tell how many they are and also to determine each of them once its position in the order is given. Then, to generate such objects uniformly at random, we can simply pick an integer uniformly at random between 1 and their total number, returning then the corresponding object. We now show how this argument can be made precise in the case of p -relations.

2.1.1 The ranking for formal languages

Suppose that \preceq is a total order relation over some finite alphabet Σ , then this order can be extended to a total order relation \preceq_{LEX} over the elements of Σ^* , called *lexicographic order*: if $\omega, \omega' \in \Sigma^*$, then $\omega \preceq_{\text{LEX}} \omega'$ iff either $\omega\sigma = \omega'$, or $\omega = \gamma a \sigma$, $\omega' = \gamma b \sigma'$ with $a \preceq b$ and $a \neq b$ (where $\gamma, \sigma, \sigma' \in \Sigma^*$ and $a, b \in \Sigma$). Hence, we have the following

Definition 2.1.1 *The rank r_L of a formal language L is the function $r_L : \Sigma^* \rightarrow \mathbf{N}$ defined as $r_L(\omega) = \#\{\omega' : \omega' \in L, |\omega'| < |\omega| \text{ or } |\omega'| = |\omega| \text{ and } \omega' \preceq_{\text{LEX}} \omega\}$; conversely, the unrank u_L is a function $u_L : \mathbf{N} \rightarrow \Sigma^*$ such that $u_L(n) = \omega$, where ω satisfies $r_L(\omega) = n$ and for every $\omega' \preceq_{\text{LEX}} \omega$, $r_L(\omega') < n$ (here we assume $|L| = \infty$).*

The complexity of computing the rank function has been studied in (Goldberg and Sipser, 1991) where it has been considered as a special kind of optimal compression: ranking is in fact related to the more general problem of storing and retrieving strings efficiently, a task very similar to the one we sketched at the beginning of this section. As one can verify, the ranking of every language in \mathbf{P} belongs to $\sharp\mathbf{P}$; in particular, several languages in \mathbf{P} are very hard to rank. This is the case, for example, of languages accepted by nondeterministic log-time bounded Turing machines, log-space bounded deterministic Turing machines, one-way nondeterministic log-space bounded Turing machines, uniform families of constant depth and polynomial size unbounded fan-in circuits, CRCW P-RAM working in constant time with a polynomial number of processor, two-way deterministic checking stack automata, two-way deterministic pushdown automata and one-way two-head deterministic finite state automata. For all these classes of languages, as proved by Huynh (1990), the ranking is even $\sharp\mathbf{P}$ -hard.

Nonetheless, classes of languages for which it is possible to efficiently compute the rank function are known. For regular languages, the ranking is NC^1 -reducible to integer division (Bertoni, Bruschi, and Goldwurm, 1991a; Huynh, 1991), and hence it can be solved by log-space uniform boolean circuits of $O(\log n \log \log n)$ depth and polynomial size (Cook, 1985). Moreover, for every language accepted by one-way unambiguous log-space bounded Turing machines, the rank function belongs to the class DET (Goldberg and Sipser, 1991) of all functions NC^1 -reducible to computing the determinant of an integer matrix (Cook, 1985); this result has been further extended to one-way log-space bounded Turing machines with bounded ambiguity degree by Bertoni and Goldwurm (1993), and similar results for Turing machines with simultaneous complexity bounds are given by Bertoni et al. (1994). Finally, Huynh (1990) showed that the rank function is in NC^2 for all languages accepted in polynomial time by one-way unambiguous (log-space) auxiliary pushdown automata hence, in particular, for unambiguous context-free languages.

2.1.2 The case of relations

We now extend the above-mentioned definition of ranking, usually given for formal languages, to the case of relations.

Definition 2.1.2 *The rank r_R of a relation $R \subseteq \Sigma^* \times \Sigma^*$ is the function $r_R : \Sigma^* \times \Sigma^* \rightarrow \mathbf{N}$ defined as $r_R(\alpha, \beta) = \#\{\beta' : \alpha R \beta', |\beta'| < |\beta| \text{ or } |\beta'| = |\beta| \text{ and } \beta' \preceq_{\text{LEX}} \beta\}$; conversely, the unrank u_R is the function $u_R : \Sigma^* \times \mathbf{N} \rightarrow \Sigma^*$ such that $u_R(\alpha, n) = \beta$, where β satisfies $r_R(\alpha, \beta) = n$ and for every $\beta' \preceq_{\text{LEX}} \beta$, $r_R(\alpha, \beta') < n$ whenever $n \leq \#\{\beta : \alpha R \beta\}$, else $u_R(\alpha, n)$ is undefined.*

Observe that the notion of rank for languages and the one here given for relations are strictly related. Given a relation $R \subseteq \Sigma^* \times \Sigma^*$, define the “corresponding” language as

$$L_R = \{\alpha \diamond \beta : \alpha R \beta\},$$

where \diamond is some distinct symbol not in Σ such that $\sigma \preccurlyeq \diamond$ for every $\sigma \in \Sigma$; conversely, given a language $L \subseteq \Sigma^*$, define the “corresponding” slice relation¹ as

$$\alpha R_L \beta \text{ iff } \alpha = a^n, \beta \in L \cap \Sigma^n, n \in \mathbf{N},$$

for some $a \in \Sigma$. It is then possible to check that if R is a p -relation, the following equalities hold

$$r_L(\alpha) = r_{R_L}(\diamond^{|\alpha|}, \alpha) + \sum_{i=1}^{|\alpha|-1} r_{R_L}(\diamond^i, \diamond^i), \quad (2.1)$$

$$r_{R_L}(\diamond^n, \alpha) = r_L(\alpha) - \max_{|\gamma| < n} r_L(\gamma), \quad (2.2)$$

$$r_R(\alpha, \beta) = r_{L_R}(\alpha \diamond \beta) - \max_{\gamma \preccurlyeq_{\text{LEX}} \alpha} r_{L_R}(\gamma \diamond \diamond^{p(|\gamma|)}), \quad (2.3)$$

$$r_{L_R}(\alpha \diamond \beta) = r_R(\alpha, \beta) + \sum_{\gamma \preccurlyeq_{\text{LEX}} \alpha} r_R(\gamma, \diamond^{p(|\gamma|)}). \quad (2.4)$$

In particular, it is possible to prove that

Lemma 2.1.1 *If $R \subseteq \Sigma^* \times \Sigma^*$ is a p -relation, then $r_L \in \text{FP}$ iff $r_{R_L} \in \text{FP}$ and if $r_{L_R} \in \text{FP}$, then $r_R \in \text{FP}$.*

Proof . The proof follows from equations (2.1)–(2.3), since the right hand side, in the given hypotheses, can be computed in polynomial time. Observe that, for equation (2.4), the sum extends over an exponential number of items, so the lemma holds in the stated direction. ■

Given these simple relationships, one can in principle directly apply the known result about the ranking for formal languages to the case of relations. Nonetheless, in the next two sections, we will give some further independent result, strictly concerning the case of p -relations.

2.1.3 The unranking

If the ranking is easy for some language L , in general it is not also true that the unranking is similarly easy. Let, for instance, L be the unary language $\{a^{2^n} : n \in \mathbf{N}\}$: it is evident that even if the ranking amounts simply to the computation of the a logarithm in base 2, the unranking function (even if its input is encoded on a unary alphabet) has an exponentially long output, thus it cannot at all be computed in polynomial time!

The situation, with p -relations, is different, since the codomain of such a relation, for every element of the domain, has a cardinality that is bounded above by a function the size of the element itself. Hence

Lemma 2.1.2 *If $R \subseteq \Sigma^* \times \Sigma^*$ is a p -relation and $r_R \in \text{FP}$, then $u_R \in \text{FP}$.*

Proof . Since $\#\{\beta : \alpha R \beta\} \leq 2^{p(|\alpha|)}$, u_L can be computed by binary search with at most $O(p(|\alpha|))$ calls to r_R . ■

¹see Section 1.2.

2.1.4 A uniform random generator

We are now able to state formally the claim sketched at the beginning of this section. Recall that in Section 1.1.3 we have designed an algorithm on PrRAM able to generate (with high probability) a number in $\{1, \dots, N\}$ uniformly at random in time $O(\log N \log \log N)$ (see Proposition 1.1.4). It is straightforward then to conclude that

Theorem 2.1.1 *If $R \subseteq \Sigma^* \times \Sigma^*$ is a p -relation and $r_R \in \text{FP}$, then R admits a polynomial time uniform random generator.*

In the next two sections of this chapter we show two classes of p -relations, characterized in terms of acceptor devices, that admit “efficient” (i.e., polynomial time) ranking. Thanks to the last theorem of this section, for both such classes it will indeed be possible to design “efficient” uniform random generators.

2.2 Turing Machines with Simultaneous Complexity Bounds

In this section, we show how it is possible to extend to the case of p -relations the result about the ranking of languages accepted by nondeterministic Turing machines with simultaneous bounds on their resources of Bertoni et al. (1994).

2.2.1 The case of formal languages

We begin by briefly recalling the model of Turing machine we refer to in this section (for a more detailed description, see (Garey and Johnson, 1979)).

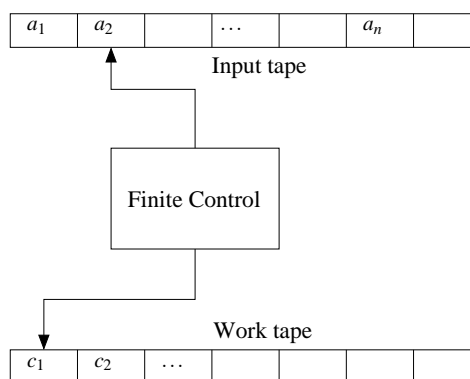


Figure 2.1: A Turing machine.

A (deterministic) *Turing machine* (TM) consists of a read-only *input tape*, a read-write *output tape* and a *finite control* (Figure 2.1). The tapes are sequences of cells each of which can hold a *tape symbol* of some alphabet Σ ; the finite control can be in one of a finite set of *states*, two of which are distinguished and called the *initial state* and *accepting state*. Given the symbols under the input and work tape heads and the current state a *next-move function* determines the next state, the head movements on both tapes and possibly a symbol to be written on the work tape. The *semantic* of the computation of the machine on input $\omega = a_1 \dots a_n$ can be defined as follows:

initially the input tape contains $a_1 \dots a_n$ all work tape cells are blank and both tape heads scan the first (leftmost) cell of the tapes. The input ω is said to be *accepted* iff the TM, started in the initial state, makes a sequence of moves which eventually enters the accepting state. The *language accepted* by a TM is the set accepted inputs.

A *nondeterministic* Turing machine (NTM) is simply a Turing machine in which the next-move function, instead of being a (partial) map from the tuple “state, input tape symbol, work tape symbol” to the tuple “state, input tape move, work tape move and symbol to write” as in the deterministic case, becomes a map from the same domain to subsets of the codomain. The computation can then take “nondeterministically”, at every step, one of the possible next moves given by such mapping. In this case a string is said to be accepted if at least one of such nondeterministic computations ends in an accepting state.

Various computational resources for such models can be defined as follows, according to the usual worst case criterion; observe that we assume without loss of generality that all the computations eventually halt. We say that a (deterministic) Turing machine works in *time* $t(n)$ (respectively, uses *space* $s(n)$, or makes $i(n)$ *inversions*) iff, for every input $a_1, \dots, a_n \in \Sigma$, the computation of the machine makes at most $t(n)$ moves (respectively, consumes at most $s(n)$ cells of the work tape, or has the input tape head change its direction at most $i(n)$ times). For the nondeterministic case, the time, space and inversion resources are similarly defined by taking the *smallest* amount of each resource consumed on every input by one of the possible nondeterministic computations; moreover, the *ambiguity* $d(n)$ is defined as the maximum number of accepting computations, taken over all the inputs $a_1, \dots, a_n \in \Sigma$.

We can now define a class of languages accepted by such machines with simultaneous bounds on the computation:

Definition 2.2.1 A language $L \subseteq \Sigma^*$ belongs to $\mathcal{B}(s(n), i(n), d(n))$, whenever it is accepted by a nondeterministic Turing machine that simultaneously uses space $s(n)$, makes $i(n)$ inversions and has ambiguity $d(n)$.

Finally, we are able to formally state the following result obtained by Bertoni et al. (1994).

Proposition 2.2.1 If a language $L \subseteq \Sigma^*$ belongs to $\mathcal{B}(s(n), i(n), d(n))$ and $s(n) \cdot i(n) \cdot d(n) = O(\log n)$, then $r_L \in \text{FP}$.

2.2.2 An extension to p -relations

In order to apply the previous result to the case of p -relations, first of all we need to extend the model of computation. The more natural way of doing so, as depicted in Figure 2.2, is to add to the standard Turing machine an additional (two-way read-only) input tape and to reserve each of the two input tapes to, respectively, the domain and codomain part of the relation.

The *semantic* of the computation of the *extended* Turing machine on input (α, β) can be defined as follows: initially the domain input tape contains $\alpha = a_1 \dots a_n$, the codomain input tape contains $\beta = b_1 \dots b_m$, all work tape cells are blank and both the tape heads scan the first (leftmost) cell of the tapes. The input (α, β) is said to be *accepted* iff the extended TM, started in the initial state, makes a sequence of moves which eventually enters the accepting state. The *relation accepted* by an extended TM is the set of ordered pairs of strings of input symbols so accepted. The semantic for the nondeterministic version can be defined similarly.

The resources of *time*, *space* and *ambiguity* for the extended model are defined as before, but *the number of input tape inversions are counted only for the codomain input tape head* (the

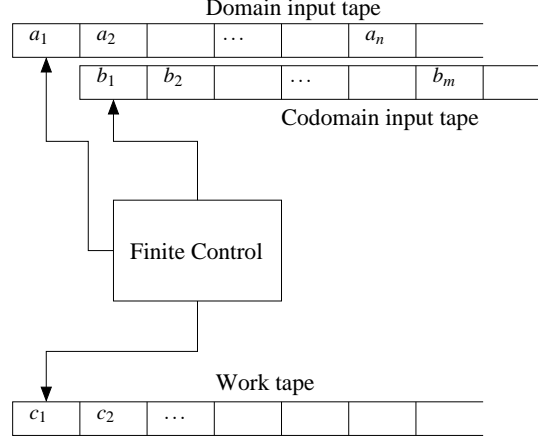


Figure 2.2: An extended Turing machine for recognizing relations.

domain input tape head has always an unrestricted number of moves). So we can finally extend Definition 2.2.1 to the case of relations, with a little abuse of notation.

Definition 2.2.2 A relation $R \subseteq \Sigma^* \times \Sigma^*$ belongs to $\mathcal{B}(s(n), i(n), d(n))$, whenever it is accepted by an extended nondeterministic Turing machine which simultaneously uses space $s(n)$, makes $i(n)$ inversions and has ambiguity $d(n)$.

We can finally give our first result on “efficiently” rankable relations. By adapting a proof of Bertoni et al. (1994), we can indeed state the following

Theorem 2.2.1 If a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ belongs to $\mathcal{B}(s(n), i(n), d(n))$ and $s(n) \cdot i(n) \cdot d(n) = O(\log n)$, then $r_R \in \text{FP}$.

Proof . Note that every TM accepting a language in $\mathcal{B}(s(n), i(n), d(n))$ can be efficiently simulated by one-way (i.e., with no input tape head inversions) TM using space $O(s(n) \cdot i(n))$ of the same order of ambiguity (Bertoni et al., 1994). It is then easy to check that the same holds for the extended model.

Moreover, if M is an extended TM accepting a p -relation R , for every $\alpha \in \Sigma^*$ and $\beta \in \Sigma^{p(|\alpha|)}$ it is immediate to construct a one-way extended TM $M^{(\beta)}$, of the same order of space and ambiguity, accepting the p -relation $R^{(\beta)}$ satisfying $\alpha R^{(\beta)} \gamma$ iff $\alpha R \gamma$ and $\gamma \preceq_{\text{LEX}} \beta$.

Hence, to prove the statement of the theorem, it is enough to show that if R is a p -relation accepted by some one-way extended TM using space $s(n)$ and having ambiguity $d(n)$ such that $s(n) \cdot d(n) = O(\log n)$, then the cardinality of $\{\gamma : \alpha R^{(\beta)} \gamma\} = r_R(\alpha, \beta)$ can be computed in a time polynomial in $|\alpha|$.

It is not hard to see that $r_R(\alpha, \beta)$ is exactly the number of $\gamma \in \Sigma^{p(|\alpha|)}$ for which $M^{(\beta)}$, having α on the first input tape and γ on the second, halts in an accepting state.

Since both input tapes are read-only, for every $\alpha \in \Sigma^n$, $\beta \in \Sigma^{p(n)}$ and $n \in \mathbb{N}$, the configurations of $M^{(\beta)}$ having α on the first input tape are at most $2^{O(s(n))} = n^{O(1)}$. Then (for instance by simulating $M^{(\beta)}$ with α on the first input tape) one can efficiently build two matrices $C_a^{(\alpha, \beta)}$, for $a \in \Sigma$, of polynomial size and values in Σ , such that the (i, j) th entry of $C_a^{(\alpha, \beta)}$ is 1 iff $M^{(\beta)}$ moves from the i th to j th configuration having α on the first input tape and reading a on the second input tape.

If π and η are, respectively, the characteristic vector of then initial and accepting configurations of $M^{(\beta)}$, then one can verify that $\pi C_{c_1}^{(\alpha, \beta)} C_{c_2}^{(\alpha, \beta)} \cdots C_{c_{p(n)}}^{(\alpha, \beta)} \eta \leq d(n)$ is the number of accepting computations of $M^{(\beta)}$ on input (α, γ) where $\gamma = c_1 c_2 \cdots c_{p(n)}$. Here we have implicitly assumed that $M^{(\beta)}$ always moves the head on its second input tape, but this is not a restriction since stationary moves can be eliminated (Bertoni and Goldwurm, 1993) by computing the transitive closure of suitable transition matrices representing stationary moves of $M^{(\beta)}$.

Let now q be an integer polynomial of degree $d(n)$ such that $q(0) = 0$ and $q(\ell) = 1$ for $1 \leq \ell \leq d(n)$; then, it is possible to check that

$$r_R(\alpha, \beta) = \sum_{i=1}^{p(n)} \sum_{c_1, c_2, \dots, c_{p(n)} \in \Sigma} q(\pi C_{c_1}^{(\alpha, \beta)} C_{c_2}^{(\alpha, \beta)} \cdots C_{c_{p(n)}}^{(\alpha, \beta)} \eta).$$

The above computation is essentially the same performed as the final step of Proposition 3 in (Bertoni and Goldwurm, 1993) and, as shown in the same paper, it can be transformed, using the matrix algebra given by direct sum and Kronecker product, in a form suitable to be computed by boolean circuits of polynomial size and depth $O(\log^2 n)$, hence, in overall polynomial time. ■

From this result, by immediate application of Theorem 2.1.1, we can introduce, via extended Turing machines with simultaneous complexity bounds, a new class of p -relations admitting a polynomial time uniform random generation

Corollary 2.2.1 *If a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ belongs to $\mathcal{B}(s(n), i(n), d(n))$ and $s(n) \cdot i(n) \cdot d(n) = O(\log n)$, then it admits a polynomial time uniform random generator.*

2.3 One-way Auxiliary Pushdown Automata

In this section, we show how to extend to the case of p -relation the result about the ranking of languages accepted by auxiliary pushdown automata by Huynh (1990).

2.3.1 The case of formal languages

Also in this case, we begin by briefly recalling the model of auxiliary pushdown automata we refer to in this section (for a more detailed description, see (Hopcroft and Ullman, 1979)).

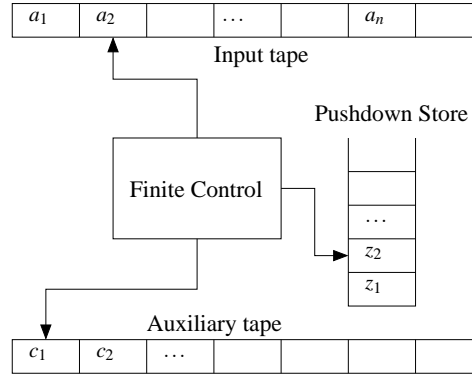


Figure 2.3: An auxiliary pushdown automaton.

A (deterministic) *one-way auxiliary pushdown automaton* (1-AuxPDA) consists of a one-way read-only *input tape*, a two-way read-write *auxiliary tape*², a *pushdown store* and a *finite control* (Figure 2.3). The tapes and the store are sequences of cells each of which can hold a *symbol* of some alphabet Σ ; one of these symbols is distinguished and called the *initial pushdown symbol*. The finite control can be in one of a finite set of *states*, two of which are distinguished and called the *initial state* and *accepting state*. Given the symbols under the input and auxiliary tape heads, the symbol on top of the pushdown store and the current state, a *next-move function* determines the next state, the auxiliary tape head movement with possibly a symbol to be written on it and an action regarding the store which can either be a push of a symbol on top, or a pop of a symbol from the top of the store. The *semantic* of the computation of the automaton on input $\omega = a_1 \dots a_n$ can be defined as follows: initially the input tape contains $a_1 \dots a_n$, the pushdown store contains the initial symbol and all auxiliary tape cells are blank and both tape heads scan the first (leftmost) cell of the tapes. The input ω is said to be *accepted* iff the automaton, started in the initial state, makes a sequence of moves which eventually enters the accepting state, with the work tape empty and the pushdown store containing only the initial symbol. The *language accepted* by such an automaton is the set of accepted strings.

²as will be stated in the following, the auxiliary tape has a logarithmic bound on the number of usable cells.

A *nondeterministic* one-way auxiliary pushdown automaton (1-NAuxPDA) is simply a one-way auxiliary pushdown automaton in which the next-move function, instead of being a (partial) map from the tuple “state, input tape symbol, work tape symbol and top of the pushdown store symbol” to the tuple “state, auxiliary tape move and symbol to write on, action on the pushdown store” as in the deterministic case, becomes a map from the same domain to subsets of the codomain. The computation can then take “nondeterministically”, at every step, one of the possible next-moves given by such mapping. In this case a string is said to be accepted if at least one of such nondeterministic computations ends in an accepting state.

Similarly to what happens with TM, various computational resources for such models can be defined, according to the usual worst case criterion; observe that we assume without loss of generality that all the computations eventually halt. We say that a (deterministic) one-way auxiliary pushdown automaton works in *time* $t(n)$ (respectively, uses *space* $s(n)$) iff, for every input $a_1 \dots a_n \in \Sigma^*$, the computation of the automaton makes at most $t(n)$ moves (respectively, consumes at most $s(n)$ cells of the auxiliary tape). For the nondeterministic case, the time and space resources are similarly defined by taking the *smallest* amount of each resource consumed on every input by one of the possible nondeterministic computations; moreover, the *ambiguity* $d(n)$ is defined as the maximum number of accepting computations, taken over all the inputs $a_1 \dots a_n \in \Sigma^*$.

Once we have defined the space resource, which measures only the space on the auxiliary tape, not of the pushdown store, we *restrict the above definitions of 1-AuxPDA and 1-NAuxPDA to automata which use at most logarithmic space*³, i.e., for which $s(n) = O(\log n)$. Moreover, the one-way *unambiguous* auxiliary pushdown automata (1-UAuxPDA) is a 1-NAuxPDA for which $d(n) = 1$, i.e., an automaton which has at most one accepting computation for every accepted string.

Finally, we can formally state the following result of Huynh (1990).

Proposition 2.3.1 *If a language $L \subseteq \Sigma^*$ is accepted by a polynomial time one-way unambiguous auxiliary pushdown automaton, then $r_L \in \text{FP}$.*

Notice that the class of languages accepted by polynomial time one-way nondeterministic auxiliary pushdown automata is exactly the class of languages that are reducible to context-free languages via one-way log-space reductions (Buntrock and Loryś, 1992).

2.3.2 An extension to p -relations

Again, in order to apply the previous result to the case of p -relations, we need to extend the model of computation. As in the case of Turing machines, the more natural way of doing so is to add to the standard auxiliary pushdown automaton an additional two-way read-only input tape for the *domain* part of the relation, leaving the usual one-way input tape for the *codomain* part (Figure 2.2).

The *semantic* of the computation of the *extended* one-way auxiliary pushdown automaton on input (α, β) can be defined as follows: initially the domain input tape contains $\alpha = a_1 \dots a_n$, the codomain input tape contains $\beta = b_1 \dots b_m$, the pushdown store contains the initial symbol, all auxiliary tape cells are blank and all tape heads scan the first (leftmost) cell of the tapes.

³without such restriction, as one can verify, such devices become in fact (computationally) equivalent to standard Turing machines.

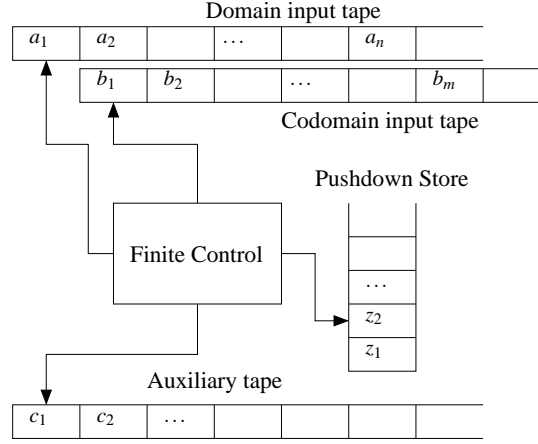


Figure 2.4: An extended auxiliary pushdown automaton for recognizing relations.

The input (α, β) is said to be *accepted* iff if the extended 1-AuxPDA, started in the initial state, makes a sequence of moves which eventually enters the accepting state. The *relation accepted* by an extended 1-AuxPDA is the set of ordered pairs of strings of input symbols so accepted. The semantic for the nondeterministic version can be similarly defined.

The resources of *time*, *space* and *ambiguity* for the extended model are defined as before, but *only the codomain input tape is restricted to be one-way* (while the domain input tape is an unrestricted two-way tape).

We are ready to give our second result on “efficiently” rankable relations. By adapting a proof of Bertoni et al. (1994), we can prove the following

Theorem 2.3.1 *If a p -relation R is accepted by a polynomial time extended one-way unambiguous auxiliary pushdown automaton, then $r_R \in \text{FP}$.*

Proof . Given an extended 1-UAuxPDA A , with state set Q , and some $\alpha \in \Sigma^*$, it is always possible to build (in time polynomial in $|\alpha|$) a 1-UAuxPDA $A^{(\alpha)}$, with state set $Q \times \{1, \dots, |\alpha|\} \times \Sigma$, such that the computation of $A^{(\alpha)}$ on input β efficiently simulates the computation of A on input (α, β) for every $\beta \in \Sigma^*$. This is made possible by using the state (q, i, a_i) of $A^{(\alpha)}$ to represent the automaton A being in state q having the two-way head on position i reading symbol a_i (where $\alpha = a_1 \dots a_n$ for some $n \in \mathbb{N}$). It is then clear that if $L^{(\alpha)}$ is the language accepted by $A^{(\alpha)}$, $r_{L^{(\alpha)}}(\beta) = r_R(\alpha, \beta)$.

As it is shown in (Huynh, 1990, Theorem 1.1), for every language L accepted by a polynomial time 1-UAuxPDA, $r_L(\beta)$ can be computed in $|\beta|^{O(1)}$ time for every $\beta \in \Sigma^*$ by means of a sequential algorithm working on a suitably defined *path system* depending on the *surface configurations* of the automaton (Cook, 1970). In order to use this result on $A^{(\alpha)}$ we need a more accurate analysis of the constants hidden in the asymptotics.

As one can verify, the (sequential) algorithm of (Huynh, 1990, Theorem 1.1), applied on a 1-UAuxPDA A' , works on input $\beta \in \Sigma^*$ in $kN^4q(|\beta|)$ time for some constant k and a polynomial q , depending on the running time of A' , where N is the number of nodes of the path system built by the algorithm. As an immediate consequence of the particular construction of the path system, $N = S^2q(|\beta|)$, where S is the number of surface configurations of A' . Recall that a surface configuration (Cook, 1970) of a 1-UAuxPDA is a tuple $(z, a, i, \gamma q \delta)$ where: z is the

topmost symbol of the stack, a is the symbol on the input tape under the head that is in position i , $\gamma\delta$ is the content of the working tape and q is the status of the automaton. Thus, $S = h|Q'| |\beta|$, where h is a constant depending on the cardinality of the alphabets (which here are assumed to be constant) and Q' is the set of states of A' .

We are then able to conclude that, since $A^{(\alpha)}$ has $2|Q| \log |\alpha|$ states and R is a p -relation, $r_{L^{(\alpha)}}(\beta)$ can be computed in time $k(2h|Q|p(|\alpha|) \log |\alpha|)^8 q^5(p(|\alpha|)) = |\alpha|^{O(1)}$. ■

Also in this case, by immediate application of Theorem 2.1.1, we can introduce, via extended one-way auxiliary pushdown automata, a new class of p -relations admitting a polynomial time uniform random generation.

Corollary 2.3.1 *If a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ is accepted by a polynomial time extended one-way unambiguous auxiliary pushdown automaton, then it admits a polynomial time uniform random generator.*

CHAPTER 3

AMBIGUOUS DESCRIPTIONS

Vous vous étonnez comme cette matière, brouillée pêle-mêle, au gré du hasard, peut avoir constitué un homme, vu qu'il avait tant de choses nécessaires à la construction de son être, mais vous ne savez pas que cent millions de fois cette matière, s'acheminant au dessein d'un homme, s'est arrêtée à former tantôt une pierre, tantôt du plomb, tantôt du corail, tantôt une fleur, tantôt une comète, pur le trop ou le trop peu de certains figures qu'il fallait ou ne fallait pas à désigner un homme?

Cyrano de Bergerac, *Voyage dans la lune*

In this chapter we discuss a new approach to uniform random generation based on the notion of ambiguous description. For the sake of simplicity, here we restrict our attention to the case of combinatorial structures instead of p -relations. After a brief restatement of the notions of uniform random generator and approximation scheme for this case, we give a formal definition of a description of a combinatorial structure, together with its ambiguity function. Then we give some general results relating combinatorial structures admitting (possibly ambiguous) description with uniform random generation, approximate and exact counting. Finally, we discuss two very simple applications of this paradigm.

3.1 Combinatorial Structures

We start this section by recalling some notions widely used in literature (Flajolet, 1988). A *combinatorial structure* is a pair $\langle \mathcal{S}, |\cdot| \rangle$, where the *domain* \mathcal{S} is a finite or denumerable set and the *size* $|\cdot| : \mathcal{S} \rightarrow \mathbf{N}$ is a function such that $\#\{s \in \mathcal{S} : |s| = n\}$ is finite for every $n \in \mathbf{N}$. Here, we implicitly assume the elements $s \in \mathcal{S}$ admit a (recursive) binary representation such that each $|s|$ is polynomially related to the length of its binary representation; this allows our model of computation to manipulate the elements of combinatorial structures. The *census function* $C_{\mathcal{S}}$ of a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ is the function $C_{\mathcal{S}} : \mathbf{N} \rightarrow \mathbf{N}$ such that $C_{\mathcal{S}}(n) = \#\{s \in \mathcal{S} : |s| = n\}$ for

every $n \in \mathbf{N}$. In the following, for the sake of brevity, we denote by \mathcal{S}_n the subset of the domain of a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ defined as $\mathcal{S}_n = \{s \in \mathcal{S} : |s| = n\} \subseteq \mathcal{S}$ (so that $C_{\mathcal{S}}(n) = \#\mathcal{S}_n$).

Different kinds of combinatorial structures are usually studied in literature, for instance: families of graphs, or trees, where the size can be considered as the number of vertices, or edges; formal languages, where the length can be taken as the size of words; discrete geometrical objects, such as polyominoes (Delest and Viennot, 1984), or tilings, where the size can be considered as the number of faces, or elementary objects constituting the whole figure.

By adapting to our case analogous definitions of Jerrum et al. (1986) recalled in Section 1.3, we now introduce the concepts of *uniform random generator* and *randomized exact counter* for a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ together with the notion of *randomized approximation scheme* for the census function $C_{\mathcal{S}}$. From now onwards we assume \perp to be a distinguished element not belonging to the domain of any combinatorial structure.

3.1.1 Uniform random generator

Definition 3.1.1 *An algorithm A is a uniform random generator (u.r.g.) for a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ iff, for every $n > 0$ such that $C_{\mathcal{S}}(n) > 0$,*

- (i) *A on input n gives output $A(n) \in \mathcal{S}_n \cup \{\perp\}$,*
- (ii) *$\Pr\{A(n) = s \mid A(n) \neq \perp\} = 1/C_{\mathcal{S}}(n)$, for every $s \in \mathcal{S}_n$, and*
- (iii) *$\Pr\{A(n) = \perp\} < 1/4$.*

Observe that the constant $1/4$ in the previous definition can be replaced by any positive number strictly less than 1 leaving the definition substantially unchanged in the sense of the following

Lemma 3.1.1 *Given a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$, let A be an algorithm for which (i) and (ii) of Definition 3.1.1 hold and such that $\Pr\{A(n) = \perp\} < \delta$, for some $0 < \delta < 1$. Then, for every $0 < \delta' < 1$ there exists an algorithm A' for which (i) and (ii) of Definition 3.1.1 hold and such that $\Pr\{A'(n) = \perp\} < \delta'$. Moreover, if A works in $T_A(n)$ time and uses $R_A(n)$ random bits, then A' works in $O(T_A(n))$ time and uses $O(R_A(n))$ random bits.*

Proof . If $\delta \leq \delta'$ the statement is trivial. Otherwise, let A' be algorithm 3.1.

```

input  $n$ 
 $s \leftarrow \perp, i \leftarrow 0$ 
while  $i < \lceil \log \delta' / \log \delta \rceil$  and  $s = \perp$  do
     $i \leftarrow i + 1$ 
     $s \leftarrow A(n)$ 
output  $s$ .
```

Algorithm 3.1: An algorithm to increase the success probability of a u.r.g..

To prove the correctness¹ of A' , let n be such that $C_S(n) > 0$, then

$$\Pr\{A'(n) = \perp\} = \Pr\{A(n) = \perp\}^{\lceil \log \delta' / \log \delta \rceil} < \delta^{\lceil \log \delta' / \log \delta \rceil} < \delta';$$

on the other hand, as it is easy to verify, $\Pr\{A'(n) = s \mid A'(n) \neq \perp\} = \Pr\{A(n) = s \mid A(n) \neq \perp\}$ for every $s \in \mathcal{S}_n$. Finally, the statements about the computation time and number of random bits used by A' follow immediately from its definition. ■

3.1.2 Randomized approximation scheme

Following Stockmeyer (1983); Karp et al. (1989), we now introduce the notion of approximate counting for combinatorial structures.

Definition 3.1.2 *An algorithm A is a randomized approximation scheme (r.a.s.) for the census function C_S of a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ iff, for every $n > 0$ such that $C_S(n) > 0$ and every $\varepsilon \in (0, 1)$,*

- (i) A on input n, ε gives output $A(n, \varepsilon) \in \mathbf{Q} \cup \{\perp\}$,
- (ii) $\Pr\{(1 - \varepsilon)C_S(n) \leq A(n, \varepsilon) \leq (1 + \varepsilon)C_S(n) \mid A(n, \varepsilon) \neq \perp\} > 3/4$ and
- (iii) $\Pr\{A(n, \varepsilon) = \perp\} < 1/4$.

Moreover, a r.a.s. is said to be a fully polynomial time r.a.s. whenever it works in time polynomial in n and $1/\varepsilon$.

Observe that even in this case the constant $1/4$ of point (iii) can be replaced by any positive number strictly less than 1 by essentially the same argument of Lemma 3.1.1; moreover, following an idea of Jerrum et al. (1986), the constant $3/4$ of point (ii) can be replaced by any number strictly between $1/2$ and 1 again leaving the definition substantially unchanged in the sense of the following

Lemma 3.1.2 *Given a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$, let A be an algorithm for which (i) and (iii) of Definition 3.1.2 hold and such that $\Pr\{(1 - \varepsilon)C_S(n) \leq A(n, \varepsilon) \leq (1 + \varepsilon)C_S(n) \mid A(n, \varepsilon) \neq \perp\} > 1/2 + \delta$, for some $0 < \delta < 1/2$. Then, for every $0 < \delta' < 1$, there exists an algorithm A' for which (i) and (iii) of Definition 3.1.2 hold and such that $\Pr\{(1 - \varepsilon)C_S(n) \leq A'(n, \varepsilon) \leq (1 + \varepsilon)C_S(n) \mid A'(n, \varepsilon) \neq \perp\} > 1 - \delta'$. Moreover, if A works in $T_A(n, \varepsilon)$ time and uses $R_A(n, \varepsilon)$ random bits, then A' works in $O(T_A(n, \varepsilon))$ time and uses $O(R_A(n, \varepsilon))$ random bits.*

Proof . If $\delta' \geq 1/2 - \delta$ the statement is trivial. Otherwise, let A' be Algorithm 3.2, where $\kappa > 0$ is an integer constant whose value will be determined in the following.

The statements about computation time and number of random bits of A' follow immediately from its definition; we prove in detail only the correctness² of A' . Fix n such that $C_S(n) > 0$ and let J be the random variable representing the value of j at the end of the execution. It is easy to observe that

$$\Pr\{A'(n, \varepsilon) = \perp\} = \Pr\{J = 0\} = \Pr\{A(n, \varepsilon) = \perp\}^{\kappa \lceil \delta^{-2} \log \delta'^{-1} \rceil} < 1/4.$$

¹for a detailed discussion of the probabilistic aspects of such proof, see Appendix A.2.2.

²for a detailed discussion of the probabilistic aspects of such proof, see Appendix A.2.2.

```

input  $n, \varepsilon$ 
 $i \leftarrow 0, j \leftarrow 0$ 
while  $i < \kappa \lceil \delta^{-2} \log \delta'^{-1} \rceil$  do
     $i \leftarrow i + 1$ 
     $c \leftarrow A(n, \varepsilon)$ 
    if  $c \neq \perp$  then
         $j \leftarrow j + 1$ 
         $c_j \leftarrow c$ 
if  $j > 0$  then
    output the median of  $c_1, \dots, c_j$ 
else
    output  $\perp$ .

```

Algorithm 3.2: An algorithm to increase the probability of correct approximation of a r.a.s..

Consider now the case $J > 0$. Let C_1, \dots, C_J be the random variables representing the values assumed by c_1, \dots, c_J at the end of the execution and let M be the random variable corresponding to the median of C_1, \dots, C_J . It is then evident that if $M \notin I(n, \varepsilon) = [(1 - \varepsilon)C_S(n), (1 + \varepsilon)C_S(n)]$, then at most one half of the C_i 's with $1 \leq i \leq J$ are such that $C_i \in I(n, \varepsilon)$. Hence, by Lemma A.2.1 and for every $k > 0$, $\Pr\{M \notin I(n, \varepsilon) \mid J = k\} \leq e^{-2k\delta^2}$ where $\Pr\{C_i \in I(n, \varepsilon)\} = \Pr\{A(n, \varepsilon) \in I(n, \varepsilon) \mid A(n, \varepsilon) \neq \perp\} > 1/2 + \delta$ by the definition of A . Hence, if $N = \kappa \lceil \delta^{-2} \log \delta'^{-1} \rceil$,

$$\begin{aligned}
 \Pr\{A'(n, \varepsilon) \notin I(n, \varepsilon) \mid A'(n, \varepsilon) \neq \perp\} &= \frac{\Pr\{M \notin I(n, \varepsilon) \cap J > 0\}}{\Pr\{J > 0\}} \\
 &= \frac{\sum_{k=1}^N \Pr\{M \notin I(n, \varepsilon) \mid J = k\} \Pr\{J = k\}}{\Pr\{J > 0\}} \\
 &\leq \frac{4}{3} \sum_{k=1}^N e^{-2k\delta^2} \binom{N}{k} q^k (1-q)^{N-k} \\
 &\leq \frac{4}{3} \sum_{k=1}^N \binom{N}{k} \left(\frac{q}{e^{2\delta^2}}\right)^k (1-q)^{N-k} \\
 &< \frac{4}{3} \left(1 - q \left(1 - \frac{1}{e^{2\delta^2}}\right)\right)^N \\
 &\leq \frac{4}{3} \left(1 - \frac{3}{4}\delta^2\right)^{\kappa \lceil \delta^{-2} \log \delta'^{-1} \rceil}
 \end{aligned}$$

where $q = \Pr\{A(n, \varepsilon) \neq \perp\} > 3/4$ and the last inequality follows from the fact that $1 - e^{-x} \geq x/2$ for $x \in [0, 1]$. Finally, by Lemma A.1.1, the integer $\kappa > 0$ can be chosen such that

$$\frac{4}{3} \left(1 - \frac{3}{4}\delta^2\right)^{\kappa \lceil \delta^{-2} \log \delta'^{-1} \rceil} < \delta'. \blacksquare$$

3.1.3 Randomized exact counter

Definition 3.1.3 An algorithm A is a randomized exact counter (r.e.c.) for a combinatorial structure $\langle S, |\cdot| \rangle$ iff, for every $n > 0$ such that $C_S(n) > 0$,

- (i) A on input n gives output $A(n) \in \mathbb{N} \cup \{\perp\}$,
- (ii) $\Pr\{A(n) = C_S(n) \mid A(n) \neq \perp\} > 3/4$ and
- (iii) $\Pr\{A(n) = \perp\} < 1/4$.

Also for this definition, one can show that the choice of the constants $1/4$ and $3/4$ is not restrictive by reasoning as in Lemma 3.1.1 and 3.1.2.

3.1.4 An example: regular languages

We now give a very simple example of the definitions given in this section. Let us consider a *deterministic finite automaton* $\mathcal{A} = (Q, q_0, \delta, F)$ over a finite alphabet Σ , where Q is the set of states, $q_0 \in Q$ is the initial state, $\delta: Q \times \Sigma \rightarrow Q$ is the transition function and $F \subseteq Q$ is the family of final states (Hopcroft and Ullman, 1979). Let $L_{\mathcal{A}} \subseteq \Sigma^*$ be the language recognized by \mathcal{A} . Hence, the combinatorial structure we consider is $\langle L_{\mathcal{A}}, |\cdot| \rangle$, where $|\cdot|$ denotes the length. Our aim is to design a u.r.g. for such a structure.

The uniform random generation algorithm first computes, for every $q \in Q$ and $1 \leq \ell \leq n$, the number $C_q(\ell)$ of all words of length ℓ accepted by (Q, q, δ, F) . This computation takes $O(n^2)$ time on a RAM under logarithmic cost criterion since each $\{C_q(\ell)\}_{\ell \geq 1}$ is the sequence of coefficients of a rational generating function (Chomsky and Schützenberger, 1963). For every of these terms, the algorithm also computes the number $b_q(\ell) = \lceil \log C_q(\ell) \rceil$ of bits required to represent $\{1, \dots, C_q(\ell)\}$ which can be obtained in $O(\ell \log \ell)$ time (see Lemma A.1.3). Hence, such a precomputation phase requires $O(n^2 \log n)$ time.

Then, the algorithm executes the procedure $\text{Generate}(q_0, n)$ described by Algorithm 3.3. Here, $\kappa \in \mathbb{N}$ is a global parameter whose value will be determined in the following, while Σ and all the sets $\{(s, \sigma) \in Q \times \Sigma : \delta(q, \sigma) = s\}$, for $q \in Q$, are endowed with some total order relation \preceq .

Reasoning by induction on ℓ , it is easy to verify that if $\text{Generate}(q, \ell)$ returns an output w different from \perp , then w is uniformly distributed in the set of words of length ℓ accepted by (Q, q, δ, F) . An upper bound to the probability that $\text{Generate}(q, \ell)$ gives output \perp depends on the global parameter κ . Denoting by $e(\ell)$ the maximum of all probabilities that $\text{Generate}(q, \ell)$ gives output \perp for $q \in Q$, one can easily show that $e(1) \leq (1/2)^\kappa$ and $e(\ell) \leq (1/2)^\kappa + e(\ell - 1)$ for every $\ell > 1$. A simple induction proves $e(n) \leq n/2^\kappa$ and hence, for every constant $t > 0$, by fixing $\kappa = t + \lceil \log n \rceil$ we obtain $e(n) \leq 1/2^t$.

As far as the time complexity is concerned, if we denote by $T(\ell)$ the maximum time cost of procedure $\text{Generate}(q, \ell)$ for $q \in Q$, for every $1 \leq \ell \leq n$, one can write the recursion $T(\ell) = O(\kappa \cdot \ell) + T(\ell - 1)$ which proves that $T(n) = O(\kappa n^2) = O((t + \log n)n^2)$.

We summarize our discussion by the following

Proposition 3.1.1 *Given a deterministic finite automaton \mathcal{A} , there exists a PrRAM which, on input $n \in \mathbb{N}$, generates a word uniformly at random in $L_{\mathcal{A}} \cap \Sigma^n$ with probability $1 - 1/2^t$ working in $O(n^2(t + \log n))$ time. Hence the combinatorial structure $\langle L_{\mathcal{A}}, |\cdot| \rangle$ admits a u.r.g. working in $O(n^2 \log n)$ time.*

3.2 Ambiguous Description

We formally introduce the concept of (*ambiguous*) *description* (see Figure 3.1):

```

procedure Generate( $q, \ell$ )
 $i \leftarrow 0, r \leftarrow \perp, w \leftarrow \perp$ 
while  $i < \kappa$  and  $r = \perp$  do
     $i \leftarrow i + 1$ 
    generate  $u \in \{1, \dots, 2^{b_q(\ell)}\}$  uniformly at random
    if  $u \leq C_q(\ell)$  then  $r \leftarrow u$ 
if  $r \neq \perp$  then
    if  $n = 1$  then
        let  $a$  be the  $r$ -th symbol in the set  $\{\sigma \in \Sigma : \delta(q, \sigma) \in F\}$ 
         $w \leftarrow a$ 
    else
        choose the smallest element  $(p, a)$  in the set  $\{(s, \sigma) \in Q \times \Sigma : \delta(q, \sigma) = s\}$ 
        such that  $\sum_{(s, \sigma) \preccurlyeq (p, a)} C_s(\ell - 1) \geq r$ 
         $w_p \leftarrow \text{Generate}(p, \ell - 1)$ 
        if  $w_p \neq \perp$  then  $w \leftarrow aw_p$ 
return  $w$ .

```

Algorithm 3.3: a uniform random generator for regular languages.

Definition 3.2.1 $\langle T, |\cdot| \rangle$ is a description of $\langle S, |\cdot| \rangle$ via the function $f : T \rightarrow S$, if f is a surjective function preserving $|\cdot|$, i.e., $|f(t)| = |t|$ for every $t \in T$. The ambiguity of the description is the function $d : S \rightarrow \mathbb{N}$ defined by $d(s) = \#\{t \in T : f(t) = s\}$, for every $s \in S$. We say that the description is ambiguous if $d(s) > 1$ for some $s \in S$. Moreover, the description is said to be polynomial whenever f and d are computable in polynomial time and there exists some $D \in \mathbb{N}$ such that $d(s) = O(|s|^D)$.

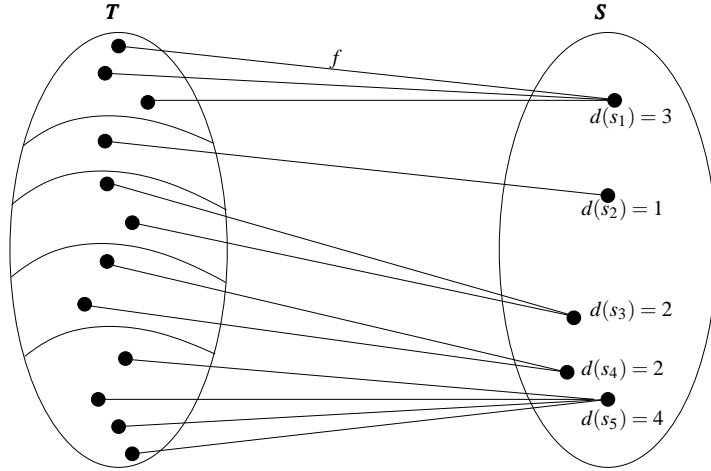


Figure 3.1: An Example of Ambiguous description.

In this thesis, we will give examples of (ambiguous) descriptions, such as the family of derivation trees of a grammar as a (possibly ambiguous) description of the strings derived in the grammar (see Section 4.1) and the family of computations of a nondeterministic device as a (possibly ambiguous) description of the strings accepted by the device (see Section 4.2).

3.2.1 Uniform random generation

We start by considering the uniform random generation problem.

Theorem 3.2.1 *If a combinatorial structure $\langle \mathcal{S}, |\cdot| \rangle$ admits a polynomial (ambiguous) description $\langle \mathcal{T}, |\cdot| \rangle$ and there exists a polynomial time u.r.g. for $\langle \mathcal{T}, |\cdot| \rangle$, then there exists a polynomial time u.r.g. for $\langle \mathcal{S}, |\cdot| \rangle$.*

To prove this theorem, we show a stronger result derived by applying the Karp, Luby, and Madras (1989) technique for sampling from a union of sets.

Lemma 3.2.1 *Let $\langle \mathcal{S}, |\cdot| \rangle$ admit a polynomial (ambiguous) description $\langle \mathcal{T}, |\cdot| \rangle$ via the function f and let $T_f(n)$ and $T_d(n)$ be respectively the computation time of f and of its ambiguity d ; moreover, assume $d(s) = O(|s|^D)$ for some $D \in \mathbf{N}$. If \mathbf{B} is a u.r.g. for $\langle \mathcal{T}, |\cdot| \rangle$ working in $T_B(n)$ time and using $R_B(n)$ random bits, then there exists a u.r.g. for $\langle \mathcal{S}, |\cdot| \rangle$ working in $O(n^{2D} + n^D(T_B(n) + T_f(n) + T_d(n)))$ time and using $O(n^{2D} + n^D R_B(n))$ random bits.*

Proof . Define, for the sake of brevity, $D(n) = \kappa_1 n^D + \kappa_2$ where κ_1, κ_2 are two integer constants such that $d(s) \leq D(|s|)$; let \mathbf{A} be Algorithm 3.4, where $\kappa > 0$ is an integer constant whose value will be determined in the following.

```

input  $n$ 
 $m \leftarrow \text{lcm}\{1, \dots, D(n)\}$ ,  $\ell \leftarrow \lceil \log m \rceil$ 
 $i \leftarrow 0$ ,  $s \leftarrow \perp$ 
while  $i < \kappa D(n)$  and  $s = \perp$  do
     $i \leftarrow i + 1$ 
     $t \leftarrow \mathbf{B}(n)$ 
    if  $t \neq \perp$  then
         $s \leftarrow f(t)$ 
        generate  $r \in \{1, \dots, 2^\ell\}$  uniformly at random
        if  $r > m/d(s)$  then
             $s \leftarrow \perp$ 
output  $s$ 

```

Algorithm 3.4: An u.r.g. for (ambiguously) described combinatorial structures.

First of all, we focus on the computation time of \mathbf{A} on input n ; the time to precompute $D(n)$, m and ℓ , by Lemma A.1.2 and A.1.3, equals $O(n^{2D})$ and the time of each iteration equals $O(T_B(n) + T_f(n) + T_d(n))$ plus the time $O(n^D)$ required for the generation of r . Moreover, each iteration uses at most $R_B(n)$ and $O(n^D)$ random bits to compute t and r respectively, hence the total amount of random bits used by \mathbf{A} adds up to $O(n^{2D} + n^D R_B(n))$.

Now, we prove the correctness of \mathbf{A} . Assume that $C_S(n) > 0$ so that, since f is surjective, $C_T(n) > 0$ and let S and T be the random variables representing respectively the value of s and t at the end of a **while** iteration (observe that S and T are well defined since their value is independent of the outcomes of the preceding iterations). Then, S takes value $s \in \mathcal{S}_n$ whenever $T \in f^{-1}(s) \subseteq \mathcal{T}_n$ and $r \leq m/d(f(T))$. Moreover, by definition of \mathbf{B} , there exists some $0 < \delta < 1/4$ such that $\Pr\{T = t\} = (1 - \delta)C_T(n)^{-1}$ for every $t \in \mathcal{T}_n$ and, as it is easy to verify since r

is independent of S and T , $\Pr\{r \leq m/d(f(T)) \mid T = t\} = d(f(t))^{-1} m 2^{-\lceil \log m \rceil}$. Hence, for every $s \in \mathcal{S}_n$,

$$\begin{aligned} \Pr\{S = s\} &= \sum_{t \in f^{-1}(s)} \Pr\{T = t, r \leq m/d(f(T))\} \\ &= \sum_{t \in f^{-1}(s)} \Pr\{r \leq m/d(f(T)) \mid T = t\} \Pr\{T = t\} \\ &= d(s) (d(s)^{-1} m 2^{-\lceil \log m \rceil} (1 - \delta) C_T(n)^{-1}) \\ &= (1 - \delta) m 2^{-\lceil \log m \rceil} C_T(n)^{-1} \end{aligned}$$

which is independent of s . On the other hand, at the end of each **while** iteration, $\Pr\{S = \perp\} = 1 - \Pr\{S \in \mathcal{S}_n\} = 1 - C_S(n) ((1 - \delta) m 2^{-\lceil \log m \rceil} C_T(n)^{-1})$. Then, since $m 2^{-\lceil \log m \rceil} > 1/2$ and $C_T(n) \leq C_S(n) D(n)$,

$$\Pr\{S = \perp\} \leq 1 - \frac{3}{8} \frac{1}{D(n)}.$$

Finally, since $\Pr\{A(n) = \perp\} = \Pr\{S = \perp\}^{\kappa D(n)}$, the integer $\kappa > 0$ can be chosen such that

$$\left(1 - \frac{3}{8} \frac{1}{D(n)}\right)^{\kappa D(n)} < 1/4$$

and hence $\Pr\{A(n) = \perp\} \leq 1/4$.

Moreover, as it is easy to verify, for every $s \in \mathcal{S}_n$, it holds $\Pr\{A(n) = s \mid A(n) = \perp\} = \Pr\{S = s \mid S \neq \perp\}$ and, since $\Pr\{S = s\}$ is constant, it is immediate to conclude that $\Pr\{A(n) = s \mid A(n) \neq \perp\} = C_S(n)^{-1}$. ■

In particular, in the case of finite ambiguity, it holds that

Corollary 3.2.1 *Under the same hypotheses as in Lemma 3.2.1, with the stronger condition $d(s) \leq D'$ for every $s \in \mathcal{S}$ and some $D' \in \mathbb{N}$, there exists a u.r.g. for $\langle \mathcal{S}, |\cdot| \rangle$ working in time $O(T_B(n) + T_f(n) + T_d(n))$ and using $O(R_B(n))$ random bits.*

3.2.2 Randomized counting

Now, let us consider the approximate and exact counting problem.

Theorem 3.2.2 *Let $\langle \mathcal{S}, |\cdot| \rangle$ be a combinatorial structure admitting a polynomial (ambiguous) description $\langle \mathcal{T}, |\cdot| \rangle$ and assume there exists a polynomial time u.r.g. for $\langle \mathcal{T}, |\cdot| \rangle$. If $C_T(n)$ is computable in polynomial time, then there exists a fully polynomial time r.a.s. for C_S . If moreover $C_T(n)$ is polynomially bounded, then there exists a polynomial time r.e.c. for $\langle \mathcal{S}, |\cdot| \rangle$.*

To prove this theorem, we show two stronger results essentially obtained by an application of Hoeffding's (1963) inequality.

Lemma 3.2.2 *Under the same hypotheses as in Lemma 3.2.1, if C_T is computable in $T_{C_T}(n)$ time, then there exists a r.a.s. for C_S working in $O(T_{C_T}(n) + n^{2D} \varepsilon^{-2} (T_B(n) + T_f(n) + T_d(n)))$ time and using $O(n^{2D} \varepsilon^{-2} R_B(n))$ random bits.*

```

input  $n$ 
 $i \leftarrow 0, j \leftarrow 0, m \leftarrow 0$ 
while  $i < \kappa \lceil D(n)/\varepsilon \rceil^2$  do
     $i \leftarrow i + 1$ 
     $t \leftarrow B(n)$ 
    if  $t \neq \perp$  then
         $j \leftarrow j + 1$ 
         $m \leftarrow m + 1/d(f(t))$ 
if  $j > 0$  then
    output  $mC_T(n)/j$ 
else
    output  $\perp$ .

```

Algorithm 3.5: A r.a.s. for ambiguously described combinatorial structures.

Proof . Define, for the sake of brevity, $D(n) = \kappa_1 n^D + \kappa_2$ where κ_1, κ_2 are two integer constants such that $d(s) \leq D(|s|)$; let A be Algorithm 3.5, where $\kappa > 0$ is an integer constant whose value will be determined in the following.

The statements about computation time and number of random bits of A follow immediately from its definition; we prove in detail only the correctness³ of A . Fix n such that $C_S(n) > 0$ and let J be the random variable representing the value of j at the end of the execution. It is easy to observe that,

$$\Pr\{A(n) = \perp\} = \Pr\{J = 0\} = \Pr\{B(n) = \perp\}^{\kappa \lceil D(n)/\varepsilon \rceil^2} < 1/4.$$

Consider now the case $J > 0$ and define for the sake of brevity

$$\begin{aligned} \mu = E(1/d(f(B(n))) \mid B(n) \neq \perp) &= \sum_{t \in T_n} 1/d(f(t)) C_T(n)^{-1} \\ &= \sum_{s \in S_n} C_T(n)^{-1} \sum_{t \in f^{-1}(s)} 1/d(s) \\ &= C_S(n) C_T(n)^{-1}. \end{aligned}$$

If M is the random variable representing the values assumed by m at the end of the execution and $I(n, \varepsilon) = [(1 - \varepsilon)\mu, (1 + \varepsilon)\mu]$, by Lemma A.2.2, $\Pr\{M/J \notin I(n, \varepsilon) \mid J = k\} \leq 2e^{-2k(\mu\varepsilon)^2}$, for

³for a detailed discussion of the probabilistic aspects of such proof, see Appendix A.2.2.

every $k > 0$. Hence, if $N = \kappa \lceil D(n)/\varepsilon \rceil^2$,

$$\begin{aligned}
 \Pr\{M/J \notin I(n, \varepsilon) \mid J > 0\} &= \frac{\Pr\{M/J \notin I(n, \varepsilon) \cap J > 0\}}{\Pr\{J > 0\}} \\
 &= \frac{\sum_{k=1}^N \Pr\{M/J \notin I(n, \varepsilon) \mid J = k\} \Pr\{J = k\}}{\Pr\{J > 0\}} \\
 &\leq \frac{8}{3} \sum_{k=1}^N e^{-2k(\mu\varepsilon)^2} \binom{N}{k} q^k (1-q)^{N-k} \\
 &\leq \frac{8}{3} \sum_{k=1}^N \binom{N}{k} \left(\frac{q}{e^{2(\mu\varepsilon)^2}}\right)^k (1-q)^{N-k} \\
 &< \frac{8}{3} \left(1 - q \left(1 - \frac{1}{e^{2(\mu\varepsilon)^2}}\right)\right)^N \\
 &\leq \frac{8}{3} \left(1 - \frac{3}{4} \left(\frac{\varepsilon}{D(n)}\right)^2\right)^{\kappa \lceil D(n)/\varepsilon \rceil^2}
 \end{aligned}$$

where $q = \Pr\{B(n) \neq \perp\} > 3/4$ and the last inequality follows from the fact that $1 - e^{-x} \geq x/2$ for $x \in [0, 1]$ and that $C_T(n) \leq C_S(n)D(n)$.

Finally, by Lemma A.1.1 the integer $\kappa > 0$ can be chosen such that

$$\frac{8}{3} \left(1 - \frac{3}{4} \left(\frac{\varepsilon}{D(n)}\right)^2\right)^{\kappa \lceil D(n)/\varepsilon \rceil^2} < \frac{1}{4}.$$

Then, since $A(n, \varepsilon) = MC_T(n)/J$, it holds

$$\Pr\{M/J \in I(n, \varepsilon) \mid J > 0\} = \Pr\{(1 - \varepsilon)C_S(n) \leq A(n, \varepsilon) \leq (1 + \varepsilon)C_S(n) \mid A(n, \varepsilon) \neq \perp\},$$

hence, $\Pr\{(1 - \varepsilon)C_S(n) \leq A(n, \varepsilon) \leq (1 + \varepsilon)C_S(n) \mid A(n, \varepsilon) \neq \perp\} \geq 3/4$. ■

We now consider the problem of obtaining a r.e.c.

Lemma 3.2.3 *Under the same hypotheses as in Lemma 3.2.2, there exists a r.e.c. for $\langle S, |\cdot| \rangle$ working in $O(T_{C_T}(n) + n^{2D}C_T(n)^2(T_B(n) + T_f(n) + T_d(n)))$ time and using $O(n^{2D}C_T(n)^2R_B(n))$ random bits.*

Proof . Using the same notation of Lemma 3.2.2, define the algorithm A' as

$$A'(n) = \text{round}\left(A\left(n, \frac{1}{3C_T(n)}\right)\right).$$

Since f is surjective $C_T(n) \geq C_S(n)$, hence A' is correct since $|A(n, \varepsilon(n))/C_S(n) - 1| < 1/(3C_T(n))$ implies $|A(n, \varepsilon(n)) - C_S(n)| < 1/3$. Finally, the statement about computation time and number of random bits used follows immediately from Lemma 3.2.2. ■

In particular, in the case of finite ambiguity, it holds that

Corollary 3.2.2 *Under the same hypotheses as in Lemma 3.2.2, with the stronger condition $d(s) \leq D'$ for every $s \in \mathbf{S}$ and some $D' \in \mathbf{N}$, there exists a r.a.s. for $C_{\mathbf{S}}$ working in $O(T_{C_{\mathbf{T}}}(n) + \varepsilon^{-2}(T_{\mathbf{B}}(n) + T_f(n) + T_d(n)))$ time and using $O(\varepsilon^{-2}R_{\mathbf{B}}(n))$ random bits and a r.e.c. for $\langle \mathbf{S}, |\cdot| \rangle$ working in $O(T_{C_{\mathbf{T}}}(n) + C_{\mathbf{T}}(n)^2(T_{\mathbf{B}}(n) + T_f(n) + T_d(n)))$ time and using $O(C_{\mathbf{T}}(n)^2R_{\mathbf{B}}(n))$ random bits.*

3.3 Simple Applications

In the remaining part of this chapter, we give two simple applications of the above results to the case of formal languages over a given alphabet Σ , i.e., for combinatorial structures $\langle \mathbf{S}, |\cdot| \rangle$ such that $\mathbf{S} \subseteq \Sigma^*$ and $|\cdot|$ is simply the word length. Given two such combinatorial structures $\langle \mathbf{S}, |\cdot| \rangle$ and $\langle \mathbf{T}, |\cdot| \rangle$, it is natural to define the *union* as $\langle \mathbf{S} \cup \mathbf{T}, |\cdot| \rangle$ and the *product* as $\langle \mathbf{S} \cdot \mathbf{T}, |\cdot| \rangle$.

We start from the uniform random generation of the product:

Theorem 3.3.1 *Let $\langle \mathbf{S}, |\cdot| \rangle$ and $\langle \mathbf{T}, |\cdot| \rangle$ be two combinatorial structures such that $\mathbf{S} \subseteq \Sigma^*$ and $\mathbf{T} \subseteq \Sigma^*$ are recognizable in polynomial time. If both structures admit a polynomial time u.r.g., then there exists a polynomial time u.r.g. for $\langle \mathbf{S} \cdot \mathbf{T}, |\cdot| \rangle$.*

Proof . If $\mathbf{P} = \mathbf{S} \times \mathbf{T}$ denotes the Cartesian product of \mathbf{S} and \mathbf{T} , then $\langle \mathbf{P}, |\cdot| \rangle$ is a polynomial (possibly ambiguous) description of $\langle \mathbf{S} \cdot \mathbf{T}, |\cdot| \rangle$ via the function $f((s,t)) = s \cdot t$, for every $(s,t) \in \mathbf{P}$; the ambiguity is defined as

$$d(q) = \sum_{s \in \mathbf{S}, t \in \mathbf{T} : s \cdot t = q} \chi_{\mathbf{S}}(s) \chi_{\mathbf{T}}(t)$$

for every $q \in \mathbf{S} \cdot \mathbf{T}$, where χ_A denotes the characteristic function of the set A . Then, if \mathbf{A} and \mathbf{B} are the polynomial time u.r.g. respectively of $\langle \mathbf{S}, |\cdot| \rangle$ and $\langle \mathbf{T}, |\cdot| \rangle$, the integer $\kappa > 0$ can be chosen such that Algorithm 3.6 is a polynomial time u.r.g. for $\langle \mathbf{P}, |\cdot| \rangle$.

```

input  $n$ 
 $i \leftarrow 0, p \leftarrow \perp$ 
while  $i < \kappa$  and  $p = \perp$  do
     $i \leftarrow i + 1$ 
     $s \leftarrow \mathbf{A}(n), t \leftarrow \mathbf{B}(n)$ 
    if  $s \neq \perp$  and  $t \neq \perp$  then
         $p \leftarrow (s, t)$ 
output  $p$ .
```

Algorithm 3.6: A u.r.g. for the product of combinatorial structures.

Therefore, by Theorem 3.2.1, there exists a polynomial time u.r.g. for $\langle \mathbf{S} \cdot \mathbf{T}, |\cdot| \rangle$. ■

Now we consider the case of the union:

Theorem 3.3.2 *Under the same hypotheses as in Theorem 3.3.1, if $C_{\mathbf{S}}$ and $C_{\mathbf{T}}$ are computable in polynomial time, then there exist a polynomial time u.r.g. for $\langle \mathbf{S} \cup \mathbf{T}, |\cdot| \rangle$.*

Proof . If $U = S \oplus T$ denotes the disjoint union of S and T , then $\langle U, |\cdot| \rangle$ is a polynomial (possibly ambiguous) description of $\langle S \cup T, |\cdot| \rangle$ via the identity function (we denote here by f) and the ambiguity is $d = \chi_S + \chi_T$. As before, the integer $\kappa > 0$ can be chosen such that Algorithm 3.7 is a polynomial time u.r.g. for $\langle U, |\cdot| \rangle$, where A and B are the polynomial time u.r.g. respectively of $\langle S, |\cdot| \rangle$ and $\langle T, |\cdot| \rangle$.

```

input  $n$ 
 $i \leftarrow 0, \ell \leftarrow \lceil \log C_S(n) + C_T(n) \rceil, u \leftarrow \perp$ 
while  $i < \kappa$  and  $u = \perp$  do
     $i \leftarrow i + 1$ 
    generate  $r \in \{1, \dots, 2^\ell\}$  uniformly at random
    if  $r \leq C_S(n)$  then
         $u \leftarrow A(n)$ 
    else if  $r \leq C_S(n) + C_T(n)$  then
         $u \leftarrow B(n)$ 
output  $u$ .
```

Algorithm 3.7: A u.r.g. for the union of combinatorial structures.

Again, by Theorem 3.2.1, there exists a polynomial time u.r.g. for $\langle S \cup T, |\cdot| \rangle$. ■

Finally, since $C_P(n) = C_S(n)C_T(n)$ and $C_U(n) = C_S(n) + C_T(n)$, from Theorem 3.2.2 simply follows

Theorem 3.3.3 *Under the same hypotheses as in Theorem 3.3.2, there exist and a fully polynomial time r.a.s. for the census functions $C_{S \cup T}$ and $C_{T \cdot S}$.*

CHAPTER 4

APPLICATIONS TO FORMAL LANGUAGES

A questo piacere contribuisce la varietà, l'incertezza, il non veder tutto, il potersi perciò spaziare coll'immaginazione, riguardo a ciò che non si vede... È piacevolissima ancora... la vista di una moltitudine innumerabile, come delle stelle, o di persone ec. un moto molteplice, incerto, confuso, irregolare, disordinato, un ondeggiamento vago ec., che l'animo non possa determinare, né concepire definitamente e distintamente...

Giacomo Leopardi, *Zibaldone di pensieri*

In this chapter we present some nontrivial application of the general paradigm introduced in the previous chapter to some class of formal languages. In particular, we show how to obtain, under suitable hypotheses, a polynomial time uniform random generator and a fully polynomial time randomized approximation schemes for polynomially ambiguous context-free languages, languages accepted in polynomial time by nondeterministic one-way auxiliary pushdown automata of polynomial ambiguity and, finally, polynomially ambiguous rational trace languages.

4.1 Context-Free Languages

A natural example of our general paradigm is given by the family of derivation trees of a context-free grammar, considered as (possibly ambiguous) description of the corresponding language. Thus, we can design simple u.r.g. and r.a.s. of census functions for inherently ambiguous context-free languages. These procedures work in polynomial time whenever the degree of ambiguity of the associated grammar is polynomially bounded. Moreover, if such a degree is bounded by a constant, both procedures require $O(n^2 \log n)$ time on input n under logarithmic cost criterion, the same order of growth required by the best known algorithms (Flajolet, Zimmerman, and Van Cutsem, 1994; Goldwurm, 1995) for the analogous problems in the unambiguous case (once their time cost is adapted to the PrRAM model of computation).

To show this application in detail, consider a context-free grammar (c.f.) $G = \langle V, \Sigma, S, P \rangle$, where V is the set of nonterminal symbols (also called variables), Σ is the alphabet of terminals, $S \in V$ is the initial variable and P is the family of productions. We assume G in Chomsky normal form (Hopcroft and Ullman, 1979) without useless variables, i.e., every nonterminal appears in a derivation of some terminal word¹. Moreover, for every $A \in V$, let T_A be the family of derivation trees with root labelled by A deriving a word in Σ^+ . It is easy to see that there are finitely many $t \in T_S$ deriving a given $x \in \Sigma^+$; in the following, we denote by $\hat{d}_G(x)$ the number of such trees and call *ambiguity* of G the function $d_G : \mathbf{N} \rightarrow \mathbf{N}$ defined by $d_G(n) = \max_{x \in \Sigma^n} \hat{d}_G(x)$, for every $n \in \mathbf{N}$. The grammar G is said *finitely ambiguous* if there exists a $k \in \mathbf{N}$ such that $d_G(n) \leq k$ for every $n > 0$; in particular, G is said *unambiguous* if $k = 1$. On the other hand, G is said *polynomially ambiguous* if, for some polynomial $p(n)$, we have $d_G(n) \leq p(n)$ for every $n > 0$.

Clearly, our idea is to use the structure $\langle T_S, |\cdot| \rangle$ as a (possibly ambiguous) description of the language L generated by G , where, for every $t \in T_S$, $|t|$ is the length of the derived word. However, to get a u.r.g. for L and a r.a.s. for its census function, we first need two preliminary procedures: one for generating a tree of given size in T_S uniformly at random, the other for computing the degree of ambiguity $\hat{d}_G(x)$ for the words $x \in \Sigma^+$.

4.1.1 Uniform random generation of derivation trees

A u.r.g. for derivation trees can be designed by following a general approach to the uniform random generation of combinatorial structures proposed by Flajolet et al. (1994). The algorithm we obtain is similar to the procedure given by Goldwurm (1995) for generating words uniformly at random in unambiguous c.f. languages. Here, we essentially adapt that routine to our case and evaluate its time complexity with respect to our model of computation.

To fix the notation, for every $A \in V$ and $\ell \in \mathbf{N}$, let T_A^ℓ be the subset $\{t \in T_A : |t| = \ell\}$, and let $C_A(\ell)$ be its cardinality. It is well known that every sequence $\{C_A(\ell)\}_{\ell \geq 1}$ has an algebraic generating function (Chomsky and Schützenberger, 1963) and hence, applying Comtet's recurrence equation (Comtet, 1964), its first n terms $C_A(1), \dots, C_A(n)$ can be computed in $O(n^2)$ time on a RAM under logarithmic cost criterion. For each $C_A(\ell)$ with $1 \leq \ell \leq n$, the integers $b_A(\ell) = \lceil \log C_A(\ell) \rceil$ are then computed, by Lemma A.1.3, in $O(\ell \log \ell)$ time, for an overall time cost of $O(n^2 \log n)$. Observe that this precomputations has to be executed only once even if we have to generate several derivation trees in T_A^ℓ .

Then, to generate a tree uniformly at random in T_S^n , we apply Algorithm 4.1 which works on input $(A, \ell) \in V \times \mathbf{N}$ and uses a global parameter κ (depending on n but not on ℓ), whose value will be determined in the following. In the procedure, P_A denotes the subset of productions in P of the form $A \rightarrow BC$ with $B, C \in V$, and P_A^1 denotes the set of productions in P of the form $A \rightarrow a$ with $a \in \Sigma$.

The procedure chooses an element uniformly at random either in P_A^1 (if $\ell = 1$), or in the set $P_A \times \{1, \dots, \ell - 1\}$ (if $\ell > 1$), the choice depending on a total order relation \leq among the elements of these sets. To define \leq , we assume a lexicographic order \preceq_{LEX} in both alphabets Σ and V and set $A \rightarrow a \leq A \rightarrow b$ in P_A^1 if $a \preceq_{\text{LEX}} b$, while $(A \rightarrow BC, h) \leq (A \rightarrow DE, k)$ in $P_A \times \{1, \dots, \ell - 1\}$ if either $h < k$, or $h = k$ and $BC \preceq_{\text{LEX}} DE$.

Reasoning by induction on ℓ , it is easy to verify that, if the output $A(\ell)$ of $\text{RandomTree}(A, \ell)$ is different from \perp , then $A(\ell)$ is uniformly distributed in T_A^ℓ . More precisely, for every $t \in T_A^n$,

¹it is well known that every c.f. language not containing the empty word ϵ can be generated by such a grammar.

```

procedure RandomTree( $A, \ell$ )
 $i \leftarrow 0, r \leftarrow \perp, t \leftarrow \perp$ 
while  $i < \kappa$  and  $r = \perp$  do
     $i \leftarrow i + 1$ 
    generate  $u \in \{1, \dots, 2^{b_A(\ell)}\}$  uniformly at random
    if  $u \leq C_A(\ell)$  then  $r \leftarrow u$ 
if  $r \neq \perp$  then
    if  $\ell = 1$  then
        let  $A \rightarrow a$  be the  $r$ -th element of  $P_A^1$ 
         $t \leftarrow (A, a)$ 
    else
        compute the smallest element  $(A \rightarrow BC, k)$  in  $P_A \times \{1, \dots, \ell - 1\}$ 
        such that  $\sum_{(A \rightarrow DE, h) \leq (A \rightarrow BC, k)} C_D(h) C_E(\ell - h) \geq r$  (*)
         $t_B \leftarrow \text{RandomTree}(B, k)$ 
         $t_C \leftarrow \text{RandomTree}(C, \ell - k)$ 
        if  $t_B \neq \perp$  and  $t_C \neq \perp$  then  $t \leftarrow (A, t_B, t_C)$ 
return  $t$ .

```

Algorithm 4.1: a uniform random generator of derivation trees.

we have

$$\Pr\{A(n) = t \mid A(n) \neq \perp\} = 1/C_A(n).$$

Now, let us give an upper bound to $\Pr\{A(n) = \perp\}$. Clearly, this value depends on the global parameter κ used by the procedure. Denoting by $e(\ell)$ the maximum of all values $\Pr\{A(\ell) = \perp\}$ for $A \in V$, one can easily show that $e(1) \leq (1/2)^\kappa$ and $e(\ell) \leq (1/2)^\kappa + \max_{1 \leq k \leq \ell-1} \{e(k) + e(\ell - k)\}$ for every $1 \leq \ell \leq n$. A simple induction proves $e(n) \leq (2n - 1)/2^\kappa$ and hence fixing $\kappa = 3 + \lceil \log n \rceil$ we have

$$\Pr\{A(n) = \perp\} \leq e(n) < 1/4 \quad \text{for every } A \in V.$$

As far as the time complexity is concerned, we assume to search the element $(A \rightarrow BC, k)$ of step $(*)$ by a boustrophedonic² routine (Flajolet et al., 1994). Again, let $N(\ell)$ be the maximum number of PrRAM instructions executed by $\text{RandomTree}(A, \ell)$ for $A \in V$. Then, for a suitable constant $c > 0$, one can write the following recursion, for every $1 \leq \ell \leq n$,

$$N(\ell) = \begin{cases} O(\kappa) & \text{if } \ell = 1 \\ O(\kappa) + \max_{1 \leq j \leq \ell} \{c \min\{j, \ell - j\} + N(j) + N(\ell - j)\} & \text{if } \ell > 1. \end{cases}$$

This proves that $N(n) = O(\kappa n) + cf(n)$, $f(n)$ being the solution of the minimax equation $f(n) = \max\{f(k) + f(n - k) + \min\{k, n - k\}\}$ usually arising in the evaluation of the cost of boustrophedonic search (Greene and Knuth, 1981). Since it is known that $f(n) = O(n \log n)$, assuming $\kappa = O(\log n)$, we get $N(n) = O(n \log n)$ which, under our model of computation, gives a total time cost $O(n^2 \log n)$, since all integers involved in the calls of this routine have $O(n)$ bits. Finally, as it is straightforward to check, the number of random bits used by the procedure is $O(n^2 \log n)$.

We observe that, following an idea described by Goldwurm (1995), the computation of the coefficients $\{(C_A(\ell), b_A(\ell)) : A \in V, 1 \leq \ell \leq n\}$ and the actual process of generating a tree uniformly at random in T_S^n can be mixed together into a unique procedure which only requires space $O(n)$ under logarithmic cost criterion (leaving unchanged the order of growth of the time complexity).

We summarize the result of this section by the following

Proposition 4.1.1 *Given a context-free grammar $G = \langle V, \Sigma, S, P \rangle$ in Chomsky normal form, there exists a u.r.g. for $\langle T_S, |\cdot| \rangle$ working in $O(n^2 \log n)$ time and using $O(n^2 \log n)$ random bits.*

4.1.2 Earley's algorithm for counting derivations

The number of derivation trees of a terminal string in a c.f. grammar can be computed by adapting Earley's algorithm (Earley, 1970) for context-free recognition. The main advantage of this procedure, with respect to the well known CYK algorithm (Harrison, 1978), is that in the case of a grammar with bounded ambiguity, the computation only requires quadratic time on a RAM under unit cost criterion (Earley, 1970; Aho and Ullman, 1972).

Let again $G = \langle V, \Sigma, S, P \rangle$ be a c.f. grammar in Chomsky normal form without useless variables. In passing, we note that the algorithm can be easily modified to work on every c.f. grammar without ϵ and unit productions. Our algorithm manipulates a *weighted* version of the so called *dotted productions* of G , i.e., expressions of the form $A \rightarrow \alpha \cdot \beta$, where $A \in V$, $\alpha, \beta \in (\Sigma \cup V)^*$ and $A \rightarrow \alpha\beta \in P$.

Given an input string $x = a_1 a_2 \dots a_n$, the algorithm computes a table of entries $S_{i,j}$, for $0 \leq i \leq j \leq n$, each of which is a list of term of the form $[A \rightarrow \alpha \cdot \beta, t]$, where $A \rightarrow \alpha \cdot \beta$ is a dotted production in G and t is a positive integer. Each pair $[A \rightarrow \alpha \cdot \beta, t]$ is called *state* and t is the *weight* of the state.

The table of lists $S_{i,j}$ computed by the algorithm has the following properties for every pair of indices $0 \leq i \leq j \leq n$:

1. $S_{i,j}$ contains at most one state $[A \rightarrow \alpha \cdot \beta, t]$ for every dotted production $A \rightarrow \alpha \cdot \beta$ in G ;

²i.e., turning like oxen in ploughing (Webster).

2. a state $[A \rightarrow \alpha \cdot \beta, t]$ belongs to $S_{i,j}$ iff there exists $\delta \in V^*$ such that $S \xRightarrow{*} a_1 \dots a_i A \delta$ and $\alpha \xRightarrow{*} a_{i+1} \dots a_j$;
3. if $[A \rightarrow \alpha \cdot \beta, t]$ belongs to $S_{i,j}$, then $t = \#\{\alpha \xRightarrow{*} a_{i+1} \dots a_j\}$, i.e., the number of leftmost derivations $\alpha \xRightarrow{*} a_{i+1} \dots a_j$.

Note that, since there are no ϵ -productions, $[A \rightarrow \alpha \cdot \beta, t] \in S_{i,i}$ implies $\alpha = \epsilon$ for every $0 \leq i \leq n$. Furthermore, once the lists $S_{i,j}$ are completed for every $0 \leq i \leq j \leq n$, the number of parse trees deriving x can be obtained by the sum $\sum_{[S \rightarrow AB, t] \in S_{0,n}} t$.

The algorithm first computes the list $S_{0,0}$ of all the states $[A \rightarrow \cdot \alpha, 1]$ such that $S \xRightarrow{*} A \delta$ for some $\delta \in V^*$. Then, it executes, for $1 \leq j \leq n$, the cycle of *Scanner*, *Predictor* and *Completer* loops given in Algorithm 4.2 computing, at the j -th loop, the lists $S_{i,j}$ for $0 \leq i \leq j$. To this end the procedure maintains a family of sets $L_{B,i}$ for $B \in V$ and $1 \leq i \leq j$; each $L_{B,i}$ contains all indices $k \leq i$ such that a state of the form $[A \rightarrow \alpha \cdot B \beta, t]$ belongs to $S_{k,i}$ for some $A \in V$, $\alpha, \beta \in V \cup \{\epsilon\}$, $t \in \mathbb{N}$. Moreover, during the computation every state in $S_{i,j}$ can be unmarked or marked according to whether it can still be used to add new states in the table.

The statement “ADD D TO $S_{i,j}$ ” simply appends the state D as unmarked to $S_{i,j}$ and updates $L_{B,i}$ whenever D is of the form $[A \rightarrow \alpha \cdot B \beta, t]$; the statement “UPDATE $[A \rightarrow \alpha \cdot \beta, t]$ IN $S_{i,j}$ ” replaces the state $[A \rightarrow \alpha \cdot \beta, u]$ in $S_{i,j}$ for some $u \in \mathbb{N}$ by $[A \rightarrow \alpha \cdot \beta, t]$, finally, “MARK D IN $S_{i,j}$ ” transforms the state D in $S_{i,j}$ into a marked state.

We are then able to prove the following

Lemma 4.1.1 *The table of lists $S_{i,j}$, $0 \leq i \leq j \leq n$, computed by the procedure described above satisfies the properties 1), 2) and 3).*

Proof . First, observe that statement 1) is easily verified: at line 5 distinct states are added to an initially empty list $S_{i,j}$; at lines 13, 18, 22 a state is added to a list provided no state with the same dotted production already appears in the list.

Statement 2) only refers to dotted productions appearing in the lists and does not concern the weight of the states. Moreover, disregarding the computations on the weight of the states, the procedure works on the dotted production exactly like Earley’s algorithm; hence statement 2) is a consequence of its correctness (for a detailed proof see (Aho and Ullman, 1972, Theorem 4.9)).

Now, let us prove statement 3). First note that all states in $S_{i,j}$, for $1 \leq i \leq j \leq n$, are marked during the computation. Hence, we can reason by induction on the order of marking states. The initial condition is satisfied because all states in each $S_{j,j}$, $0 \leq j \leq n$, are of the form $[A \rightarrow \cdot \alpha, t]$ and have weight $t = 1$. Also the states of the form $[A \rightarrow a \cdot, t]$ have weight $t = 1$ and again statement 3) is satisfied.

Then, consider a state $D = [A \rightarrow \alpha \cdot B \beta, w] \in S_{k,j}$ with $k < j$. We claim that w is the number of leftmost derivations $\alpha B \xRightarrow{*} a_{k+1} \dots a_j$. A state of this form is first added by the *Completer* at line 13. This means that there exists a set of indices I_k such that for every $i \in I_k$ there is $[A \rightarrow \alpha \cdot B \beta, u_i] \in S_{k,i}$ with $u_i \in \mathbb{N}$, and a family U_i of sates $[B \rightarrow \gamma \cdot, t] \in S_{i,k}$ such that

$$w = \sum_{i \in I_k} \sum_{[B \rightarrow \gamma \cdot, t] \in U_i} t u_i.$$

Observe that $k \leq i < j$ for every $i \in I_k$ and U_i is the subset of all states in $S_{i,j}$ with a dotted production of the form $B \rightarrow \gamma \cdot$. Moreover, each state $[A \rightarrow \alpha \cdot B \beta, u_i] \in S_{k,i}$ is marked at line 16

1 **for** $j = 1 \dots n$ **do**

Scanner:

2 **for** $i = j - 1 \dots 0$ **do**
 3 **for** $[A \rightarrow \cdot a, t] \in S_{i,j-1}$ **do**
 4 MARK $[A \rightarrow \cdot a, t]$ IN $S_{i,j-1}$
 5 **if** $a = a_j$ **then** ADD $[A \rightarrow a \cdot, t]$ TO $S_{i,j}$

Completer:

6 **for** $i = j - 1 \dots 0$ **do**
 7 **for** $[B \rightarrow \gamma \cdot, t] \in S_{i,j}$ **do**
 8 MARK $[B \rightarrow \gamma \cdot, t]$ IN $S_{i,j}$
 9 **for** $k \in L_{B,i}$ **do**
 10 **for** $[A \rightarrow \alpha \cdot B\beta, u] \in S_{k,i}$ **do**
 11 **if** $[A \rightarrow \alpha B \cdot \beta, v] \in S_{k,j}$
 12 **then** UPDATE $[A \rightarrow \alpha B \cdot \beta, v + tu]$ IN $S_{k,j}$
 13 **else** ADD $[A \rightarrow \alpha B \cdot \beta, tu]$ TO $S_{k,j}$

Predictor:

14 **for** $i = 0 \dots j - 1$ **do**
 15 **for** $[A \rightarrow \alpha \cdot B\beta, t] \in S_{i,j}$ **do**
 16 MARK $[A \rightarrow \alpha \cdot B\beta, t]$ IN $S_{i,j}$
 17 **for** $B \rightarrow \gamma \in P$ **do**
 18 **if** $[B \rightarrow \cdot \gamma, 1] \notin S_{j,j}$ **then** ADD $[B \rightarrow \cdot \gamma, 1]$ TO $S_{j,j}$
 19 **while** \exists UNMARKED $[A \rightarrow \cdot B\beta, t] \in S_{j,j}$ **do**
 20 MARK $[A \rightarrow \cdot B\beta, t]$ IN $S_{j,j}$
 21 **for** $B \rightarrow \gamma \in P$ **do**
 22 **if** $[B \rightarrow \cdot \gamma, 1] \notin S_{j,j}$ **then** ADD $[B \rightarrow \cdot \gamma, 1]$ TO $S_{j,j}$

Algorithm 4.2: Part of Earley's algorithm modified to count derivation trees.

or 20 before D is added to $S_{k,j}$. Also all states in U_i , for all $i \in I_k$, are marked during the computation of the weight w . Observe that, due to the form of the grammar, updating such a weight w cannot modify the weight of any state in U_i . As a consequence all the states in U_i are marked before D . Hence, by inductive hypothesis, we have for every $i \in I_k$

$$u_i = \#\{\alpha \xRightarrow{*} a_{k+1} \dots a_i\} \quad (4.1)$$

and, for each $[B \rightarrow \gamma \cdot, t] \in U_i$,

$$t = \#\{\gamma \xRightarrow{*} a_{i+1} \dots a_j\}. \quad (4.2)$$

Now the number of leftmost derivations $\alpha B \xRightarrow{*} a_{k+1} \dots a_j$ is clearly given by

$$\sum_{k \leq i < j} \#\{\alpha \xRightarrow{*} a_{k+1} \dots a_i\} \sum_{B \rightarrow \gamma \in P} \#\{\gamma \xRightarrow{*} a_{i+1} \dots a_j\}$$

and the claim follows from statement 1) and equalities (4.1) and (4.2). ■

Theorem 4.1.1 *Given a context-free grammar G in Chomsky normal form and assuming the RAM model under logarithmic cost criterion, the algorithm described above computes the number of derivation trees of an input string of length n in $O(n^4)$ time, consuming $O(n^3)$ space. If the grammar G is finitely ambiguous, then the algorithm has time complexity $O(n^2 \log n)$ and space complexity $O(n^2)$.*

Proof. We first observe that every list $S_{i,j}$ for $0 \leq i \leq j \leq n$ contains at most $O(1)$ states, each of which can be represented by one integer of size respectively $O(1)$ or $O(n)$ according to whether G is finitely ambiguous or not. Since the space taken by the algorithm is essentially due to the memory required by the table $S_{i,j}$, we obtain a space complexity $O(n^2)$ for finitely ambiguous grammars and $O(n^3)$ in the general case.

As far as the time complexity is concerned, note that in each loop, for a fixed $1 \leq j \leq n$, the *Scanner* and *Predictor* phase execute $O(j)$ statements, while the *Completer* phase requires $O(j\ell)$ unit steps, where ℓ is the maximum size of the sets $L_{B,i}$, $B \in V$, $0 \leq i \leq j-1$. Now, for a general grammar, we have $\ell = O(j)$ which implies a total number of unit steps $O(n^3)$: each of them requires logarithmic time $O(n)$ leading the overall time complexity to $O(n^4)$.

On the contrary, in the case of a finitely ambiguous grammar G , we have $\ell = O(1)$ and hence we only need $O(\log n)$ logarithmic time to locate each state in the table yielding a total time complexity $O(n^2 \log n)$. ■

4.1.3 Inherently ambiguous context-free languages

Now, let us go back to our original problem and let $L \subseteq \Sigma^*$ be a c.f. language. We recall that L is unambiguous if it is generated by an unambiguous c.f. grammar, while it is *inherently ambiguous* whenever every c.f. grammar G generating L is ambiguous. We also say that L is *finitely* (respectively, *polynomially*) *ambiguous* if it is generated by a finitely (respectively polynomially) ambiguous c.f. grammar.

Observe that there are natural examples of polynomially ambiguous c.f. languages which are not finitely ambiguous. For instance, if $L = \{ww^R : w \in \{a,b\}^*\}$, then it turns out that L^2 is inherently ambiguous, but not finitely ambiguous (Harrison, 1978, Section 7.3): however, it is easy to verify that L^2 is generated by a grammar of ambiguity $O(n)$ given by the set of productions $S \rightarrow AB$, $A \rightarrow aAa|bAb|\epsilon$ and $B \rightarrow aBa|bBb|\epsilon$. Similarly, for every $k \in \mathbb{N}$, the language $(L^2 \cdot \{\diamond\})^k$ is generated by a grammar of ambiguity $O(n^k)$.

Then, applying Propositions 4.1.1 and Theorem 4.1.1 to Theorems 3.2.1 and 3.2.2, we obtain

Theorem 4.1.2 *If L is a polynomially ambiguous context-free language, then there exists a polynomial time u.r.g. for L and a fully polynomial time r.a.s. for its census function C_L . If moreover C_L is polynomially bounded, then there exists a polynomial time r.e.c. for L .*

In particular, in the case of finite ambiguity, one gets the following

Corollary 4.1.1 *If L is a finitely ambiguous context-free language, then there exists a u.r.g. for L working in $O(n^2 \log n)$ time on a PrRAM under logarithmic cost criterion and a r.a.s. for its census function C_L of the same time complexity.*

4.1.4 The grammar as a part of the input

Observe that the results of Sections 4.1.1 and 4.1.2 assume as fixed a c.f. grammar G . In view of an application presented in the next section, here we study the complexity of similar algorithms taking G as a part of the input.

To this end, we need a definition of the dimension of a c.f. grammar; if $G = \langle V, \Sigma, S, P \rangle$, we define its dimension $|G|$ as the number of bits needed to encode it, which is $O(\#P\ell_G \log(\#V + \#\Sigma))$, where ℓ_G is the maximum number of symbols appearing in the right hand side of a production in P . Also in this case, we restrict to c.f. grammars in Chomsky normal form, since it is known (Harrison, 1978) that every c.f. grammar G can be transformed in a c.f. grammar G' in Chomsky normal form in time polynomial in $|G|$ (and hence $|G'|$ is polynomially related to $|G|$).

Given a c.f. grammar $G = \langle V, \Sigma, S, P \rangle$ (in Chomsky normal form) and $n > 0$, what is essentially needed to generate (uniformly at random) a derivation tree of a word of length n , are the coefficients $C_A(\ell)$, for $1 \leq \ell \leq n$ and $A \in V$ (see Section 4.1.1). Observe that the method used in that section, fundamentally based on Comtet's recurrence equation, can, in general, require time exponential in $|G|$. Nonetheless, by a modified version of the CYK algorithm (Harrison, 1978), we obtain the following

Lemma 4.1.2 *There exists an algorithm that, having as input a c.f. grammar $G = \langle V, \Sigma, S, P \rangle$ (in Chomsky normal form), $A \in V$ and $n > 0$, computes $C_A(n)$ in $O(n^3|G|)$ time on a RAM under logarithmic cost criterion.*

Proof . We design such an algorithm by an application of dynamic programming and of the well known convolution property

$$C_A(\ell) = \sum_{A \rightarrow BC \in P} \sum_{i=1}^{\ell-1} C_B(i) C_C(\ell-i)$$

which holds for every $A \in V$ and $\ell > 0$. Consider Algorithm 4.3, where $c(A; k)$, for $A \in V$ and $1 \leq k \leq n$, is an array of integer.

```

1  input  $G = \langle V, \Sigma, S, P \rangle, A, n$ 
2  for all  $B \in V$  do  $c(B; 1) \leftarrow 0$ 
3  for all  $B \rightarrow a \in P$  do  $c(B; 1) \leftarrow c(B; 1) + 1$ 
4  for  $k = 2 \dots n$  do
5      for all  $B \rightarrow CD \in P$  do
6           $c(B; k) \leftarrow c(B; k) + \sum_{0 < i < k} c(C; i) c(D; k-i)$ 
7  output  $c(A; n)$ .
```

Algorithm 4.3: Counting derivation trees.

To prove the correctness of the algorithm, we prove a somehow stronger result: at the beginning of each iteration of the loop at line 4, $c(B; \ell) = C_B(\ell)$ for every $B \in V$ and $1 \leq \ell < k$. By induction on k : if $k = 2$, then the loop at lines 2 and 3 compute exactly $C_A(1)$ for every $B \in V$, moreover, the loop at line 4 makes no iteration, hence the statement is true. If $k > 2$, then $\sum_{0 < i < k} c(C; i) c(D; k-i)$, by inductive hypothesis, is equal to the number of parse trees in T_B^k with the first derivation corresponding to $B \rightarrow CD$. Then the loop at line 5 collects in $c(B; k)$ the value $C_B(k)$ by iteratively summing over all the production in P with B on the left, for every

$B \in V$. To obtain an upper bound on the computation time, observe that the sum of line 6 has at most n summands and is in a nested loop which is executed $O(n\#P)$ times, and that all the involved integers have size $O(n)$ bits. ■

Then, as discussed in Section 4.1.1, we can apply Algorithms 4.3 and 4.1 to obtain the following

Proposition 4.1.2 *There exists a u.r.g. of derivation trees that, given a c.f. grammar G in Chomsky normal form and $n > 0$ as input, works on a PrRAM in time polynomial in $|G|$ and n .*

Furthermore, the algorithm of Section 4.1.2 to compute the number of derivation trees of a terminal string, can be used also assuming the grammar as part of the input. A simple analysis leads to the following

Proposition 4.1.3 *The number of derivation trees of a terminal string x in a context-free grammar G in Chomsky normal form can be computed in time polynomial in $|G|$ and $|x|$ on a RAM under logarithmic cost criterion.*

4.2 One-way Nondeterministic Auxiliary Pushdown Automata

In this section we show how to apply the general technique of Chapter 3 to the case of languages accepted by polynomial time *one-way nondeterministic auxiliary pushdown automata* (1-NAuxPDA, see Section 2.3 for a definition of the model).

We recall that, given a 1-NAuxPDA M , the *ambiguity* of M is the function $d_M : \mathbb{N} \rightarrow \mathbb{N}$ defined as the maximum number of accepting computations for every input string of length $n \in \mathbb{N}$. Hence, we say that M is *polynomially ambiguous* if, for some polynomial $p(n)$, we have $d_M(n) \leq p(n)$ for every $n > 0$. We also recall that it is well known that, given an integer input $n > 0$, a context-free grammar G_n can be built, in time polynomial in n , such that $L(G_n) \cap \Sigma^n = L(M) \cap \Sigma^n$, where $L(G_n) \subseteq \Sigma^*$ is the language generated by G_n and $L(M) \subseteq \Sigma^*$ is the language accepted by M . This allows us to apply the results of the previous section to the languages accepted by 1-NAuxPDA.

Here, we describe a modified version of the usual construction of G_n given by Cook (1971); Allender, Bruschi, and Pighizzini (1993) which allows to bound the ambiguity of G_n with respect to the ambiguity of M . First of all, we assume without loss of generality that the automaton cannot simultaneously consume input and modify the content of the stack and that at most one symbol can be pushed or popped for each single move.

A *surface configuration* (Cook, 1970) of a 1-NAuxPDA M on input $x \in \Sigma^+$ of length n is a 5-tuple (q, w, i, Γ, j) where q is the state of M , w the content of its work tape, $1 \leq i \leq |w|$ the work tape head position, Γ the symbol on top of the stack and $1 \leq j \leq n + 1$ the input tape head position. Observe that there are $n^{O(1)}$ surface configurations on any input of length $n \geq 0$. Two surface configurations C_1, C_2 form a *realizable pair* (C_1, C_2) (on a word $y \in \Sigma^+$) iff M can move (consuming input y) from C_1 to C_2 , ending with its stack at the same height as in C_1 , without popping below this height at any step of the computation. If (C_1, D) is a realizable pair on y' and (D, C_2) is a realizable pair on y'' , then it is straightforward to check that (C_1, C_2) is a realizable pair on $y = y'y''$. Let \mathcal{S}_n be the set of surface configurations of M on inputs of length n and define the c.f. grammar $G_n(M)$ by the following statements:

- (a) the set of nonterminal symbol V contains each (C_1, C_2, ℓ) such that $C_1, C_2 \in \mathcal{S}_n$ and $\ell \in \{0, 1\}$;
- (b) the set of terminal symbols is Σ ;
- (c) the initial variable is $(C_{\text{in}}, C_{\text{fin}})$, where C_{in} and C_{fin} represent respectively the initial and final surface configuration of M ;
- (d) the set P of productions is given as follows:
 - (1) $(C_1, C_2, 0) \rightarrow \sigma \in P$ iff (C_1, C_2) is a realizable pair on $\sigma \in \Sigma \cup \{\varepsilon\}$ via a single move computation;
 - (2) $(C_1, C_2, 0) \rightarrow (C_1, D, 1)(D, C_2, \ell) \in P$, for $\ell \in \{0, 1\}$, iff $C_1, C_2, D \in \mathcal{S}_n$;
 - (3) $(C_1, C_2, 1) \rightarrow (D_1, D_2, \ell) \in P$, for $\ell \in \{0, 1\}$, iff $C_1, D_1, D_2, C_2 \in \mathcal{S}_n$, D_1 can be reached from C_1 by a single move pushing a symbol on top of the stack and C_2 can be reached from D_2 by a single move popping the same symbol from the top of the stack.

For the sake of brevity define, for each realizable pair (C_1, C_2) , the set $\mathcal{C}(C_1, C_2)$ of all computations of M starting from C_1 and ending in C_2 with the stack at the same height as in C_1 , without popping below this height at any point of the computation; define also the subset $\mathcal{C}_1(C_1, C_2) \subseteq \mathcal{C}(C_1, C_2)$ of such computations (of at least two steps) during which the stack height never equals the stack height at the extremes C_1, C_2 and the subset $\mathcal{C}_0(C_1, C_2) = \mathcal{C}(C_1, C_2) \setminus \mathcal{C}_1(C_1, C_2)$ of computations during which the stack height equals, at least once, the stack height at the extremes C_1, C_2 .

Fist of all, observe that, following Cook (1971) and Allender et al. (1993), it is possible to prove the

Proposition 4.2.1 *Given a 1-NAuxPDA M working in polynomial time, there exists an algorithm computing, on input $n > 0$, the c.f. grammar $G_n(M)$ in polynomial time on a RAM under logarithmic cost criterion.*

We now turn to show that the c.f. grammar $G_n(M)$ defined above exactly generates the set of strings of length n accepted by M :

Lemma 4.2.1 *Given a 1-NAuxPDA M , the c.f. grammar $G_n(M)$ is such that*

- (i) *if (C_1, C_2) is a realizable pair on y via a computation in $\mathcal{C}_\ell(C_1, C_2)$, then $(C_1, C_2, \ell) \xRightarrow{*} y$;*
- (ii) *if $(C_1, C_2, \ell) \xRightarrow{*} y$, then (C_1, C_2) is a realizable pair on y via a computation in $\mathcal{C}_\ell(C_1, C_2)$.*

Proof . We prove statement (i) by induction on the number m of computation steps from C_1 to C_2 . If $m = 1$ then the computation can only be in $\mathcal{C}_0(C_1, C_2)$ and consumes at most one symbol $\sigma \in \Sigma \cup \{\varepsilon\}$; then, by (d1), $(C_1, C_2, 0) \rightarrow \sigma \xRightarrow{*} \sigma$.

If $m > 1$ and the computation is in $\mathcal{C}_0(C_1, C_2)$ let D be the first surface configuration after C_1 in the computation for which the stack height equals that of C_1, C_2 ; it is then straightforward to check that (C_1, D) is a realizable pair on some y' via a computation in $\mathcal{C}_1(C_1, D)$ and (C_2, D) is a realizable pair on some y'' via a computation in $\mathcal{C}_\ell(D, C_2)$, for some $\ell \in \{0, 1\}$, where $y = y'y''$; moreover these computations have both fewer steps then the one between

C_1 and C_2 . Hence, by inductive hypothesis, $(C_1, D, 1) \xRightarrow{*} y'$ and $(D, C_2, \ell) \xRightarrow{*} y''$ and, by (d2), $(C_1, C_2, 0) \rightarrow (C_1, D, 1)(D, C_2, \ell) \xRightarrow{*} y'y'' = y$.

Finally, if $m > 1$ and the computation is in $\mathcal{C}_1(C_1, C_2)$, then it is straightforward to check that the first move after C_1 (say, to a surface configuration D_1) has to be a push and the last move before C_2 (say, from a surface configuration D_2), has to be a pop of the same symbol; moreover, since the moves from C_1 to D_1 and from D_2 to C_2 alter the stack and hence consume no input, (D_1, D_2) is a realizable pair on y via a computation in $\mathcal{C}_\ell(D_1, D_2)$, for some $\ell \in \{0, 1\}$, have fewer steps than the computation between C_1 and C_2 . Then, by inductive hypothesis, $(D_1, D_2, \ell) \xRightarrow{*} y$ and, by (d3), $(C_1, C_2, 1) \rightarrow (D_1, D_2, \ell) \xRightarrow{*} y$.

Now we prove statement (ii) by induction on the number k of derivation steps. If $k = 1$ the only production leading to a terminal symbol is the one in (d1), hence the statement is obvious.

On the other hand, if $k > 1$ and $\ell = 0$, then, by (d2), the derivation can only be of the form $(C_1, C_2, 0) \rightarrow (C_1, D, 1)(D, C_2, \ell') \xRightarrow{*} y'y'' = y$, for some $\ell' \in \{0, 1\}$, where $(C_1, D, 1) \xRightarrow{*} y'$ and $(D, C_2, \ell') \xRightarrow{*} y''$ and both derivations have less than k steps; then, by inductive hypothesis, (C_1, D) is a realizable pair on y' and (D, C_2) is a realizable pair on y'' , hence (C_1, C_2) is a realizable pair on y via a computation in $\mathcal{C}_0(C_1, C_2)$.

If $k > 1$ and $\ell = 1$, then, by (d3), the derivation can only be of the form $(C_1, C_2, 1) \rightarrow (D_1, D_2, \ell') \xRightarrow{*} y$, for some $\ell' \in \{0, 1\}$, where $(D_1, D_2, \ell') \xRightarrow{*} y$ in $k - 1$ steps; by inductive hypothesis, (D_1, D_2) is a realizable pair on y ; moreover, by (d3), D_1 can be reached from C_1 with a single push move and C_2 can be reached from D_2 with a single pop of the same symbol, hence it is straightforward to verify that (C_1, C_2) is a realizable pair on y via a computation in $\mathcal{C}_1(C_1, C_2)$. ■

In order to apply the result on c.f. grammars of Section 4.1 in this case, we have to bound the ambiguity of $G_n(M)$. To this end, we have the following

Lemma 4.2.2 *Given a 1-NAuxPDA M , the number of leftmost derivations $(C_1, C_2, \ell) \xRightarrow{*} y$ in $G_n(M)$ is less than or equal to the number of computations in $\mathcal{C}_\ell(C_1, C_2)$ consuming y .*

Proof . We want to prove that, for every $C_1, C_2 \in \mathcal{S}_n$ and $y \in \Sigma^*$, the proofs of Lemma 4.2.1 define a bijective map from the leftmost derivations $(C_1, C_2, \ell) \xRightarrow{*} y$ to the computations in $\mathcal{C}_\ell(C_1, C_2)$ consuming y .

By the construction, it is clear that such a map is surjective. Hence, we simply have to show that it is injective, i.e., if the computations associated with two leftmost derivations are equal, then the two derivations themselves are the same. We work by induction on the number m of steps of the computation. If $m = 1$ the statement is obvious.

If $m > 1$ and $\ell = 0$, let, for $i \in \{1, 2\}$, $(C_1, C_2, 0) \rightarrow (C_1, D_i, 1)(D_i, C_2, \ell_i) \xRightarrow{*} y$ be two leftmost derivations (for some $\ell_i \in \{0, 1\}$ and $D_i \in \mathcal{S}_n$). By construction of the grammar, if the associated computations are equal, then $D_1 = D_2$, $\ell_1 = \ell_2$ and there exist two words y', y'' such that $y = y'y''$, $(C_1, D_i, 1) \xRightarrow{*} y'$ and $(D_i, C_2, \ell_i) \xRightarrow{*} y''$, for $i \in \{1, 2\}$. Then, by inductive hypothesis, $(C_1, D_i, 1) \xRightarrow{*} y'$ are the same leftmost derivation for $i \in \{1, 2\}$ and $(D_i, C_2, \ell_i) \xRightarrow{*} y''$ are the same leftmost derivation for $i \in \{1, 2\}$. Hence $(C_1, C_2, 0) \rightarrow (C_1, D_i, 1)(D_i, C_2, \ell_i) \xRightarrow{*} y$ are the same leftmost derivation for $i \in \{1, 2\}$.

If $m > 1$ and $\ell = 1$, let, for $i \in \{1, 2\}$, $(C_1, C_2, 1) \rightarrow (D_{1,i}, D_{2,i}, \ell_i) \xRightarrow{*} y$ be two leftmost derivations (for some $\ell_i \in \{0, 1\}$ and $D_{1,i}, D_{2,i} \in \mathcal{S}_n$). If the associated computations are equal, then

again by the construction of the grammar, $D_{1,1} = D_{1,2}$, $D_{2,1} = D_{2,2}$ and $\ell_1 = \ell_2$, hence, for $i \in \{1, 2\}$, $(D_{1,i}, D_{2,i}, \ell_i) \xRightarrow{*} y$. Moreover, by inductive hypothesis, the two derivations are the same leftmost derivation for $i \in \{1, 2\}$, hence $(C_1, C_2, 1) \rightarrow (D_{1,i}, D_{2,i}, \ell_i) \xRightarrow{*} y$ are the same leftmost derivation for $i \in \{1, 2\}$. ■

We can now conclude our discussion: by Theorem 3.2.1 and 3.2.2, an application of the results of Section 4.1.4 and of this section, leads to the following

Theorem 4.2.1 *Let L be the language accepted by a polynomial time 1-NAuxPDA with polynomial ambiguity. Then there exists a polynomial time u.r.g. for L and a fully polynomial time r.a.s. for its census function C_L . If moreover C_L is polynomially bounded, then there exists a polynomial time r.e.c. for L .*

4.3 Rational Trace Languages

Another application concerns the uniform random generation and the census function of trace languages. To study this case we refer to Diekert and Métivier (1997) for basic definitions. We only recall that, given a trace monoid $\mathbb{M}(\Sigma, I)$ over the independence alphabet (Σ, I) , a *trace language*, i.e., a subset of $\mathbb{M}(\Sigma, I)$, is usually specified by considering a string language $L \subseteq \Sigma^*$ and taking the closure $[L] = \{t \in \mathbb{M}(\Sigma, I) : t = [x] \text{ for some } x \in L\}$. In particular, a trace language $T \subseteq \mathbb{M}(\Sigma, I)$ is called *rational* if $T = [L]$ for some regular language $L \subseteq \Sigma^*$. In this case we say that T is *represented* by L and the *ambiguity* of this representation is the function $d_L : \mathbb{N} \rightarrow \mathbb{N}$, defined by $d_L(n) = \max_{x \in \Sigma^n} \#(L \cap [x])$. We say that a rational trace language T is *finitely ambiguous* if it is represented by a regular language L such that, for some $k \in \mathbb{N}$, $d_L(n) \leq k$ for every $n > 0$; in this case we say that T has *ambiguity* k .

In the following, assuming a given independence alphabet (Σ, I) , we denote by \mathcal{R} the set of all rational trace languages in $\mathbb{M}(\Sigma, I)$ and, by \mathcal{R}_k , the subset of trace languages in \mathcal{R} of ambiguity k . Clearly, for every independence alphabet (Σ, I) , we have $\mathcal{R}_1 \subseteq \mathcal{R}_2 \subseteq \dots \subseteq \bigcup_{k=1}^{\infty} \mathcal{R}_k \subseteq \mathcal{R}$.

The properties of these families of languages have been studied in the literature by Bertoni et al. (1982); Bertoni, Mauri, and Sabadini (1985); Sakarovitch (1987) extending a previous study on unambiguous rational sets in totally commutative monoids by Eilenberg and Schützenberger (1969). In particular, it is known that $\mathcal{R}_1 = \mathcal{R}$ iff the independence relation I is transitive (Bertoni et al., 1985; Sakarovitch, 1987); on the other hand, if I is not transitive, then we get the following chain of strict inclusions:

$$\mathcal{R}_1 \subsetneq \mathcal{R}_2 \subsetneq \dots \subsetneq \bigcup_{k=1}^{\infty} \mathcal{R}_k \subsetneq \mathcal{R}.$$

We say that a rational trace language T is *polynomially ambiguous* if it is represented by a regular language L such that, for some polynomial $p(n)$, we have $d_L(n) \leq p(n)$ for every $n > 0$. Observe that there exist examples of polynomially ambiguous rational trace languages that are not finitely ambiguous. For instance, fixing $I = \{(a, b), (b, c)\}$, if $L = (a^*c)^*(ab)^*c(a^*c)^*$, then it turns out that $[L]$ does not belong to $\bigcup_{k=1}^{\infty} \mathcal{R}_k$ (Bertoni et al., 1982): however, $[L]$ is polynomially ambiguous with $d_L(n) = O(n)$ since, for every x of the form $x = a^{k_1}ca^{k_2}c \dots (ab)^{k_s}ca^{k_{s+1}}c \dots a^{k_t}c$ with $k_1, \dots, k_s, \dots, k_t \in \mathbb{N}$, it holds

$$L \cap [x] = \{a^{k_1}c \dots (ab)^{k_i}c \dots a^{k_t}c : k_i = k_s, 1 \leq i \leq t\}.$$

Now, let us go back to our problem: here, we want to use L as an ambiguous description of $[L]$. Also in this case, we first have to design two routines: one for generating a word in L uniformly at random and the other for determining the number of representatives of a trace in a given regular language L . The first routine is already discussed in Section 3.1.4. The other algorithm is obtained by adapting a procedure for solving the membership problem for rational trace language (Bertoni, Mauri, and Sabadini, 1989; Avellone and Goldwurm, 1998):

Proposition 4.3.1 *Given an independence alphabet (Σ, I) and a regular language $L \subseteq \Sigma^*$, the problem of determining the cardinality of $L \cap [x]$, given $x \in L$ as input, can be solved on a RAM under unit cost criterion in $O(n^\alpha)$ time, where α is the maximum clique size in (Σ, I) .*

As a consequence, we can apply the general paradigm for uniform random generation and approximate counting presented in Chapter 3. This leads to the following

Theorem 4.3.1 *Let $T \subseteq \mathbb{M}(\Sigma, I)$ be a finitely ambiguous rational trace language and assume $I \neq \emptyset$. Then, T admits a u.r.g. working in $O(n^\alpha \log n)$ time and using $O(n^2 \log n)$ random bits on a PrRAM under logarithmic cost criterion, where α is the size of the maximum clique in (Σ, I) . Moreover, there exists a r.a.s. for the census function of T of the same time complexity. On the other hand, if T is polynomially ambiguous, then it admits a polynomial time u.r.g. and a fully polynomial time r.a.s. for its census function.*

CHAPTER 5

SOME REMARKS

*Spesso c'è bonaccia sulla pagina.
Inutile girarla per cercare
l'angolo del vento.
Si sta fermi,
il pensiero oscilla,
si riparano le cose
che la navigazione ha guastato.*

Valerio Magrelli, *Ora serrata retinae*

In this brief chapter we investigate how to extend some result of the two previous chapters about combinatorial structures back to the more general case of p -relations. In particular, we shortly investigate some closure property of some classes of p -relations under union and complement boolean operators, once suitably adapted to the case of p -relations.

5.1 From Combinatorial Structures to p -Relations

As we have stated at the beginning of Chapter 3, for the sake of brevity and clarity in the definitions and proofs, we have restricted our attention to combinatorial structures instead of p -relations. Here we want to briefly and informally suggest that the results obtained so far can be generalized back to the case of p -relations.

To this aim, one needs a definition of *description* suitable to the case of relation. A proposal can be the following: a p -relation $S \subseteq \Sigma^* \times \Sigma^*$ is an (*ambiguous*) *description* of a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ via the function $f : S \rightarrow R$ iff, for every $\alpha, \beta \in \Sigma^*$ such that $\alpha R \beta$, there exists some $\gamma \in S(\alpha)$ such that $f(\alpha, \gamma) = (\alpha, \beta)$. In this case, the *ambiguity* of the description is the function $d : R \rightarrow \mathbb{N}$ defined by $d((\alpha, \beta)) = \#\{\gamma \in S(\alpha) : f(\alpha, \gamma) = (\alpha, \beta)\}$ and the description is said to be

polynomial whenever f and d are computable in time polynomial in $|\alpha|$ and there exists some $D \in \mathbf{N}$ such that $d((\alpha, \beta)) = O(|\alpha|^D)$.

The results of Chapter 3 can be rephrased according to the following

Corollary 5.1.1 *If a p -relation R has a polynomially (ambiguous) description S which admits a polynomial time u.r.g., then R admits a polynomial time u.r.g.; moreover, if, for every $\alpha \in \Sigma^*$, the cardinality of $S(\alpha)$ can be computed in time polynomial in $|\alpha|$, then R admits a fully polynomial r.a.s..*

In particular, Theorem 2.3.1 of Section 4.2 can be generalized to extended one-way non-deterministic auxiliary pushdown automata (see Section 2.3), obtaining the following stronger version of Corollary 2.3.1

Corollary 5.1.2 *If a p -relation $R \subseteq \Sigma^* \times \Sigma^*$ is accepted by a polynomially ambiguous extended one-way nondeterministic auxiliary pushdown automaton working in polynomial time, then R admits a polynomial time uniform random generator.*

Proof . The proof follows from an argument similar to the beginning of the proof of Theorem 2.3.1: given an extended 1-NAuxPDA M , with state set Q , and some $\alpha \in \Sigma^*$, it is always possible to build (in time polynomial in $|\alpha|$) a 1-NAuxPDA $M^{(\alpha)}$, with state set $Q \times \{1, \dots, |\alpha|\} \times \Sigma$, such that the computation of $M^{(\alpha)}$ on input β efficiently simulates the computation of A on input (α, β) for every $\beta \in \Sigma^*$ and has the same ambiguity as M . The result then follows by applying Theorem 4.2.1 to $M^{(\alpha)}$. ■

5.2 The p -Union

In this section we want to investigate the closure with respect to the union of some of the classes of p -relations we have encountered in previous chapters.

First of all, we need the following

Definition 5.2.1 *Let $S \subseteq \Sigma^* \times \Sigma^*$ and $R \subseteq \Sigma^* \times \Sigma^*$ be two p -relations. Their p -union $R \cup^p S$ is defined as $\alpha (R \cup^p S) \beta$ iff $\alpha R \beta$ or $\alpha S \beta$, whenever the polynomials which relate the length of the elements in the domain to the length of the elements in the codomain are identical, otherwise is undefined.*

Then, by means of Corollary 5.1.1 and Theorem 3.3.2, one can obtain the following

Proposition 5.2.1 *Let $S \subseteq \Sigma^* \times \Sigma^*$ and $R \subseteq \Sigma^* \times \Sigma^*$ are two p -relations such that $R \cup^p S$ is defined. If $r_R \in \text{FP}$ and $r_S \in \text{FP}$, then $R \cup^p S$ admits a polynomial time u.r.g..*

On the other hand, it is known (Huynh, 1990, Theorem 3.1) that there exist two unambiguous context free languages \hat{L}_1, \hat{L}_2 such that their union is inherently ambiguous and its rank is not polynomial time computable unless $P = P^{\sharp P}$. Moreover, it is also known (Goldberg and Sipser, 1991) that unambiguous context free languages are rankable in polynomial time and hence the slice relations¹ of \hat{L}_1, \hat{L}_2 are both rankable in polynomial time.

We can summarize this discussion in the following

Proposition 5.2.2 *The class of p -relations that are rankable in polynomial time is not close under p -union, hence it is a proper subclass of the class of p -relations admitting a polynomial time u.r.g. (unless $P = P^{\sharp P}$).*

5.3 The p -Complement

Now we consider the closure with respect to the complement of some of the classes of p -relations. For this reason, we suggest the following

Definition 5.3.1 *Given a p -relation $R \subseteq \Sigma^* \times \Sigma^*$, its p -complement R^{pc} is the p -relation $R^{pc} \subseteq \Sigma^* \times \Sigma^*$ defined as $\alpha R^{pc} \beta$ iff not $\alpha R \beta$ and $|\beta| = p(|\alpha|)$.*

Now, we show that there exists a p -relation \hat{R} such that it admits a polynomial time u.r.g., but the same does not hold for its p -complement. Let $\hat{R} = (G, U)$ be the p -relation such that $U \subseteq V$ is an independent set of the graph $G = \langle V, E \rangle$, i.e., for every pair of nodes $u, v \in U$ the edge (u, v) does not belong to E ; if $V = \{v_1, \dots, v_n\}$, assume to represent U over $\{0, 1\}^*$ by a string $u_1 \dots u_n$ such that u_i is 1 iff $v_i \in U$, for $1 \leq i \leq n$.

It is then clear that \hat{R}^{pc} is the relation (G, U) where either G is not (a correct encoding of) a graph and U is any string of length $p(|G|)$, or $G = \langle V, E \rangle$ is a graph and $U \subseteq V$ is such that $(u, v) \in E$ for some $u, v \in U$.

As proved by Sinclair (1988, Theorem 1.17), if \hat{R} admits a polynomial time u.r.g., then $\text{NP} = \text{RP}$. However, using the algorithm for generating uniformly at random the satisfying assignment to a boolean formula in disjunctive normal form (DNF) given by Jerrum et al. (1986), we can prove the following

Lemma 5.3.1 *There exists a polynomial time u.r.g. for \hat{R}^{pc} .*

Proof . Let G be an element of the domain of R . If G is not a graph, it is easy to generate uniformly at random an element of $\{0, 1\}^{p(|G|)}$. On the other hand, let $G = \langle V, E \rangle$ be a graph and consider the DNF

$$\bigvee_{(v,v') \in E} x_v \wedge x_{v'}$$

¹see Section 1.2.

in the variables $X_G = \{x_v : v \in V\}$. It is easy to verify that every truth assignment $\tau : X_G \rightarrow \{\text{true}, \text{false}\}$ satisfies the DNF iff $U = \{v : \tau(x_v) = \text{true}\}$ is such that $(G, U) \in \hat{R}^{pc}$. Hence, the result follows by applying the u.r.g. for the DNF given by Jerrum et al. (1986). ■

On the other hand, by observing that $r_{R^{pc}}(\alpha, \beta) = \#\{\gamma : \gamma \preceq_{\text{LEX}} \beta\} - r_R(\alpha, \beta)$, one can immediately conclude the following

Proposition 5.3.1 *The class of p -relation that are rankable in polynomial time is closed under p -complement, hence it is a proper subclass of the class of p -relations admitting a polynomial time u.r.g. (unless $\text{NP} = \text{RP}$).*

The situation discussed in these last two section it then illustrated by Figure 5.1, where the endpoints of the dotted arrows are as shown (from top to bottom) if $\text{NP} \neq \text{RP}$ and $\text{P} \neq \text{P}^{\#P}$.

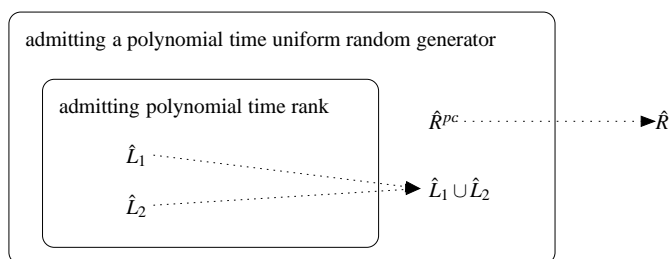


Figure 5.1: Some classes of p -relations.

CHAPTER 6

APPLICATIONS TO COMBINATORIAL OPTIMIZATION

*Hay que cansar los números.
Que cuenten sin parar,
que se embriaguen contando,
y que no sepan ya
cuál de ellos será el último:
¡qué vivir sin final!
Que un gran tropel de ceros
asalte nuestras dichas
esbeltas, al pasar,
y las lleve a su cima.
Que se rompan las cifras,
sin poder calcular
ni el tiempo ni los besos.*

Pedro Salinas, *La voz a ti debida*

In this chapter we discuss an application of uniform random generation to *combinatorial optimization*; notice that the research field of combinatorial optimization is so relevant to the theory and practice of computer science that part of the contents of this chapter has been the very inspiration originally motivating this entire work.

After a short general introduction, we focus on the *local search* paradigm which is probably the most used technique to solve combinatorial optimization problems since it combines simplicity and empirical success in many applications. As studied by Grossi (1999), one way to improve and fully exploit the power of local search consist in the use of uniform random generators for producing an initial solution; in connection to this fact, it has been possible to define the class of *expectation-guaranteed* problems: a wide and natural class whose member admit efficient deterministic approximation algorithms obtained by *derandomizing* such a process of

random generation. In the last section of this chapter we show a negative result concerning this approach by proving that not every problem admits an efficient derandomization, unless $P = \#P$.

6.1 Some Basic Definitions and Properties

This section is intended to be a very short introduction to the basic definitions and known facts about combinatorial optimization; a complete and exhaustive treatment of this subject is far behind the scope of this work; for a more comprehensive treatment of this topic, see for instance (Garey and Johnson, 1979; Bovet and Crescenzi, 1993; Papadimitriou, 1994; Hochbaum, 1997).

6.1.1 The class NPO

We begin by recalling the definition of a particular class of combinatorial optimization problems which captures the most part of problems of practical interest; we observe that our definition is slightly different from the standard one because of our use of p -relations

Definition 6.1.1 A NP optimization (NPO) problem is a 3-tuple (R, g, goal) where $R \subseteq \Sigma^* \times \Sigma^*$ is a p -relation, the objective function $g : \Sigma^* \times \Sigma^* \rightarrow \mathbf{N}$ is a polynomial time computable function and $\text{goal} \in \{\min, \max\}$. Moreover, the problem is said to be polynomially bounded whenever the value of $g(\alpha, \beta)$ is bounded by a polynomial in $|\alpha|$, for every α, β .

As we have anticipated in Section 1.2, the p -relation R serves both to express, via its domain, the set of instances of the problem $\text{dom}(R) = \{\alpha \in \Sigma^* : \alpha R \beta \text{ for some } \beta \in \Sigma^*\}$ and, for each instance $\alpha \in \text{dom}(R)$, the set of its feasible solutions $\text{sol}(\alpha) = \{\beta \in \Sigma^* : \alpha R \beta\}$. The objective function, also called *value*, or *measure* of the solution, together with the goal of the problem, defines its *optimum* as

$$\text{opt}(\alpha) = \underset{\beta \in \text{sol}(\alpha)}{\text{goal}} \ g(\alpha, \beta).$$

Hence, the *solution* of a NPO problem consists in finding, for every instance $\alpha \in \text{dom}(R)$, an *optimum solution* $\tilde{\beta} \in \text{sol}(\alpha)$ such that $g(\alpha, \tilde{\beta}) = \text{opt}(\alpha)$. Finally, recall that with every NPO problem it can be associated its *decision version* whose solution is to decide, for every pair (α, t) with $\alpha \in \text{dom}(R)$ and $t \in \mathbf{Q}$, whether $\text{opt}(\alpha) \leq t$ (if $\text{goal} = \min$), or $\text{opt}(\alpha) \geq t$ (if $\text{goal} = \max$).

Some example of problems in this class are the “Max Cut” and “Max Clique” problems¹. In both cases, the instances are graphs $G = (V, E)$. For “Max Cut” case, for each graph G , the feasible solutions are partitions of its vertex set V into disjoint sets $\langle V_1, V_2 \rangle$ and the goal is to maximize the measure of a partition, defined as the cardinality of the induced cut, i.e., the number of edges with one endpoint in V_1 and the other in V_2 . On the other hand, in “Max Clique” problem, for each graph G , the feasible solutions are subset of vertices $V' \subseteq V$ which are all pairwise adjacent, i.e., every pair of vertices in V' are joined by an edge in E ; in this case the goal is to maximize the measure of such subsets, defined as their cardinality.

¹for a thorough compendium of NPO problems, see the survey of Crescenzi and Kann (1998).

6.1.2 Approximation algorithms and approximation classes

It is well known (Karp, 1972; Garey and Johnson, 1979) that, under the conjecture $P \neq NP$, every NPO problem whose corresponding decision version is NP-complete is not solvable in polynomial time. As a consequence, if we are interested in “efficient” algorithms, we have to restrict our attention only to polynomial time computable “approximate solutions” of the problem. More formally

Definition 6.1.2 Let (R, g, goal) be a NPO problem. Given an instance $\alpha \in \text{dom}(R)$ and a feasible solution $\beta \in \text{sol}(\alpha)$, the performance ratio $\mathcal{R}(\alpha, \beta)$ of β with respect to α is

$$\mathcal{R}(\alpha, \beta) = \max \left\{ \frac{\text{opt}(\alpha)}{g(\alpha, \beta)}, \frac{g(\alpha, \beta)}{\text{opt}(\alpha)} \right\}$$

and the relative error $\mathcal{E}(\alpha, \beta)$ of β with respect to α is

$$\mathcal{E}(\alpha, \beta) = \frac{|\text{opt}(\alpha) - g(\alpha, \beta)|}{\max\{\text{opt}(\alpha), g(\alpha, \beta)\}}.$$

Obviously, the closer to 1 (respectively 0) is the performance ratio (respectively relative error), the better are the solutions. There is a strict relationship between the performance ratio and the relative error:

$$\mathcal{R}(\alpha, \beta) = \frac{1}{1 - \mathcal{E}(\alpha, \beta)}.$$

The concepts expressed so far allow us to define what an *approximation algorithm* is

Definition 6.1.3 Given a function $r : \mathbf{N} \rightarrow [1, \infty)$, an algorithm A is a $r(n)$ -approximation algorithm for a NPO problem (R, g, goal) iff, for every instance $\alpha \in \text{dom}(R)$, it returns a feasible solution $A(\alpha) \in \text{sol}(\alpha)$ such that

$$\mathcal{R}(\alpha, A(\alpha)) \leq r(|\alpha|).$$

Similarly, one can define an *approximation scheme* as follows

Definition 6.1.4 An algorithm A is an approximation scheme for a NPO problem (R, g, goal) iff, for every instance $\alpha \in \text{dom}(R)$ and $r \in (1, \infty)$, it returns a feasible solution $A(\alpha, r) \in \text{sol}(\alpha)$ such that

$$\mathcal{R}(\alpha, A(\alpha, r)) \leq r.$$

An approximation scheme is said to be *polynomial* if it works in time polynomial in $|\alpha|$ and *fully polynomial* if it works in time polynomial in $|\alpha|$ and $r/(r-1)$.

This notion gives rise in a natural way to various classes of NPO problems according to the existence of approximation algorithms satisfying specific performance bound. As an example, we just recall some of such classes we will refer to later on in this chapter. The class of NPO problems approximable within a constant factor is defined as follows

Definition 6.1.5 A NPO problem belongs to the class APX if, for some $r > 1$, it exists a polynomial time r -approximation algorithm for it.

Then we can look for problems admitting efficient approximation schemes

Definition 6.1.6 A NPO problem belongs to the class PTAS if it admits a polynomial time approximation scheme; similarly, it belongs to the class FPTAS if it admits a fully polynomial time approximation scheme.

Observe that these classes do not trivially coincide since, for instance, it is known that every problems in NPO whose decision version is strongly NP-complete cannot have a fully polynomial time approximation scheme (Garey and Johnson, 1979). As it is obvious to conclude from the previous definitions

Proposition 6.1.1 $\text{FPTAS} \subseteq \text{PTAS} \subseteq \text{APX}$.

6.1.3 Logically definable approximation classes

Other classes of NPO problems can be described by means of the so called *logical definability* (for a survey, see the work of Kolaitis and Thakur (1994)), a concept inspired by Fagin's characterization of NP in terms of second-order logic on finite structures (Fagin, 1974). More precisely, according to such approach, a class of finite structures is NP-computable iff it is definable by an *existential second-order formula*, i.e., an expression of the form $\exists \mathbf{S} \phi(\mathbf{G}, \mathbf{S})$, where \mathbf{G} is a *finite structure*, \mathbf{S} is a finite sequence of predicate symbols of bounded arity, and $\phi(\mathbf{G}, \mathbf{S})$ is a first-order formula. It is well known that every such formula is equivalent to a formula in *prenex normal form*, i.e., one of the form $\exists \mathbf{S} \forall \mathbf{x} \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{G}, \mathbf{S})$, where φ is a quantifier-free formula and $\mathbf{x} = (x_1, \dots, x_j)$, $\mathbf{y} = (y_1, \dots, y_k)$ are finite sequences of first-order variables (for an introduction to the logic details, see, for instance, (Barwise, 1977)).

According to this framework, Papadimitriou and Yannakakis (1991) introduced the class MaxNP of maximization problems whose optimum can be defined as

$$\max_{\mathbf{S}} |\{\mathbf{x} : \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{G}, \mathbf{S})\}|,$$

where φ is a quantifier-free first order formula. Intuitively, in a NP decision problem one seeks the predicates \mathbf{S} that satisfy some existential second-order sentence $\forall \mathbf{x} \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{G}, \mathbf{S})$, while in the corresponding maximization version in MaxNP one seeks those predicates \mathbf{S} that maximize the number of tuples \mathbf{x} satisfying the existential first-order sentence $\exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y}, \mathbf{G}, \mathbf{S})$.

A canonical example in such class is the “Max Sat” problem², the instances of which are collections C of clauses, i.e., a disjunction of literals, where a literal is a variable or a negated variable; feasible solutions are truth assignments to the variables and the goal is to maximize the number of clauses satisfied by the truth assignment. The optimum of such problem can be written as

$$\max_T |\{\omega \in C : \exists x [(T(x) \wedge P(\omega, x)) \vee (\neg T(x) \wedge N(\omega, x))]\}|$$

where $T = \{x \in X : \tau(x) = 1\}$ is the set of variables which are true under the truth assignment $\tau : X \rightarrow \{0, 1\}$, and the binary predicates $P(\omega, x)$ and $N(\omega, x)$ express the fact that the literal x appears as positive, respectively negative, in the clause ω .

The syntactic view has proven useful not only in obtaining structural complexity results, but also in developing paradigms for designing efficient approximation algorithms as, for instance, stated by Papadimitriou and Yannakakis (1991).

²For a more formal definition, see (Crescenzi and Kann, 1998).

Proposition 6.1.2 *For every problem in MaxNP there exists a polynomial time approximation algorithm whose performance ratio is bounded by a constant, therefore*

$$\text{MaxNP} \subseteq \text{APX}.$$

The same clearly holds for a subclass of MaxNP, called MaxSNP (“S” stands for *strict*), consisting of those maximization problems that are defined by quantifier-free formulae, such as

$$\max_{\mathbf{s}} |\{\mathbf{x} : \varphi(\mathbf{x}, \mathbf{G}, \mathbf{S})\}|,$$

where φ is quantifier-free. The class MaxSNP contains several natural maximization problems such as “Max 3-Sat” (the version of “Max Sat” with exactly three literals per clause).

6.1.4 Approximation preserving reductions

A basic notion in complexity theory is that of *reducibility*. Roughly speaking, a reduction between two decision problems allows one to transform every algorithm for solving a problem in an algorithm to solve the other. More formally, if the languages $L, L' \subseteq \Sigma^*$ encode two decision problems, we say that there is a *many-one* reduction from L to L' if there exists a polynomial time computable function $f : \Sigma^* \rightarrow \Sigma^*$ such that $\omega \in L$ iff $f(\omega) \in L'$.

Unfortunately, in the context of approximation, these reductions do not guarantee to preserve the objective function, and in general, the quality of the solution. Thus, new *approximation-preserving reductions* have been introduced, i.e., reductions that not only map instances of one problem into instances of the other, but that are also able to map back “good” solutions into “paragonably good” solutions. As an example of such idea, we only recall the definition of AP-reduction (Crescenzi, Kann, Protasi, and Trevisan, 1995), that will be used later in this chapter.

Definition 6.1.7 A NPO problem (R, g, goal) is said to be AP-reducible to a NPO problem (R', g', goal') iff there exist two polynomial time computable functions f and b , and a constant ε such that, for every $\alpha \in \text{dom}(R)$, $r > 1$ and $\beta' \in \text{sol}'(f(\alpha, r))$,

1. $f(\alpha, r) \in \text{dom}(R')$;
2. $b(\alpha, \beta', r) \in \text{sol}(\alpha)$;
3. $\mathcal{R}(f(\alpha, r), \beta') \leq r$ implies $\mathcal{R}(\alpha, b(\alpha, \beta', r)) \leq 1 + \varepsilon(r - 1)$.

Given a concept of reduction, it is possible to define the concepts of *hardness*, *completeness* and *closure* for a given approximation class \mathcal{C} of NPO problems as in the case of decision problems

Definition 6.1.8 A NPO problem is said to be \mathcal{C} -hard (under a given approximation preserving reduction) if all NPO problems in \mathcal{C} can be reduced to it; in addition, the problem is said to be \mathcal{C} -complete if it also belongs to \mathcal{C} . The closure of \mathcal{C} under such reduction is the class $\bar{\mathcal{C}}$ of all NPO problems reducible to some problem in \mathcal{C} .

These ideas are widely studied, for instance by Papadimitriou and Yannakakis (1991); Khanna, Motwani, Sud (1994). We observe that one more reason of interest for the study of the class MaxNP is that the first example of a complete problem has been given for it (Khanna et al., 1994).

Proposition 6.1.3 The NPO problem “Max Sat” is complete for MaxNP.

6.2 Local Search

In this section we introduce the basic idea of the *local search paradigm* with a discussion of some of its strengths and limits. Then we discuss in which sense randomization can help to improve such a paradigm. Here, for the sake of brevity, we focus only on maximization problems; a similar discussion can be easily carried out for the minimization case.

6.2.1 The basic idea

In this section, to simplify the notation, we restrict our attention to a very “simple version” of a combinatorial optimization problem. Let X be a discrete search space and let $f : X \rightarrow \mathbf{N}$ be the goal function that must be maximized on a set of feasible solutions $S \subseteq X$. The *local search paradigm* is a general scheme to design approximation algorithms based on stepwise improvement on the value of the function f by exploring a “neighborhood” of the current solution.

A *neighborhood structure* is a map $S \rightarrow 2^S$ that defines, for each $x \in S$, a set $S_x \subseteq S$ of solutions that are, in some sense, “close” to x (we assume the map is such that $y \in S_x$ iff $x \in S_y$). The set S_x is then called the *neighborhood* of the solution x , and each $y \in S_x$ is called *neighbor* of x .

As an example, if $X = \{0, 1\}^n$, one can define, for every $h > 0$, the so called *h-bounded neighborhood structure* as the map $x \mapsto \{y \in X : d_H(x, y) \leq h\}$, where $d_H : \mathbf{X} \times \mathbf{X} \rightarrow \mathbf{N}$ is the *Hamming distance*³.

³We recall that $d_H(x, y)$ is defined as the number of positions in which x differs from y .

Once a neighborhood structure is fixed, an example of the local search paradigm is given by the schema of Algorithm 6.1.

```

input  $x_0 \in S$ 
 $x \leftarrow x_0$ 
repeat
    pick  $y \in S_x$ 
    if  $f(y) > f(x)$  then  $x \leftarrow y$ 
until  $f(y) \leq f(x)$  for all  $y \in S_x$ 
output  $x$ .

```

Algorithm 6.1: Local Search: schema of algorithm.

With the term “schema” we refer to the fact that various methods can be used to implement the *pick* statement, selecting one neighbor of the current solution, such as choosing at random, following some predefined order, or taking the neighbor of maximum value.

The fundamental concept in the analysis of a local search algorithm is that of local, as opposed to global, optimality: $x \in S$ is a *local maximum* (with respect to a given neighborhood structure) iff $f(x) \geq f(y)$ for all $y \in S_x$ while $x \in S$ is a *global maximum* iff $f(x) \geq f(y)$ for all $y \in S$. It is then clear that one of the main drawbacks of the local search paradigm is that in general it offers algorithms that terminate in a local optimum and, unfortunately, in general it is not easy to estimate how far it is from a global one.

There is a clear trade-off between the quality of the local optima and the neighborhood structure: the larger are the neighborhoods the better are the local optima, but it may be harder (slower) to compute them. Moreover, the quality of the local optima is also strictly related to the choice of the *initial solution* x_0 ; usually, to reduce the sensibility to it, local search algorithms are repeated on many different initial solutions (and then the best output obtained is adopted). Whenever it is hard to come out with a feasible initial solution, clearly another trade-off happens between the number of initial solution considered and the quality of the final local optimum obtained.

Therefore, designing a good local search algorithm has remained up to now an experimental art, even if some guiding principles and techniques have been identified. Nonetheless, as experimentally shown, the local search paradigm usually leads to fast algorithms that find reasonably good solutions in a reasonable amount of time. For this reason, they cover a wide application area ranging from mathematical problems, to graph problems, to logical problems and so on.

6.2.2 Local search and NPO problems

Thanks to its generality and its empirical success, the local search paradigm represents certainly one of the most used approaches in approximating difficult optimization problems. The complexity of finding locally optimal solutions for those problems has been mainly investigated by Johnson, Papadimitriou, and Yannakakis (1988); Ausiello and Protasi (1995), showing the strengths and the limits of such a paradigm.

In particular, Ausiello and Protasi (1995) introduce the class GLO of NPO problems that have *guaranteed local optima*: a maximization problem (R, g, \max) has guaranteed local optima

iff there exists an integer constant h such that the value of all local optima is at least $1/h$ times the value of a global optimum. If a problem has guaranteed local optima, then there exists an h -bounded neighborhood structure such that, for every instance $\alpha \in \text{dom}(R)$, every local optimum β of α has the property that $\text{opt}(\alpha) \leq h \cdot g(\alpha, \beta)$ (a similar definition can be given when $\text{goal} = \text{min}$). The fundamental assumption here is that, given an instance α of a (polynomially bounded) NPO problem, it is possible to determine whether a given solution β of α is a local optimum, and in the negative case, to find a better solution in its neighborhood in polynomial time. Various problems⁴ can be shown to belong to GLO, such as “Max Cut”, “Max Sat”, “Max 3-Sat”. Moreover, the class GLO is a subclass of APX, but the inclusion is proper as shown for instance by the problem “Max k -GSat” which is well known to be approximable within a constant factor, but that does not have guaranteed local optima (Alimonti, 1996). A further result is that $\overline{\text{GLO}} = \text{APX}$, where the closure is taken with respect to the AP-reduction.

6.3 The “Expectation-Guaranteed” Class

Different techniques have been suggested to overcome some of the drawbacks of the local search paradigm. Some of these are, for instance, based on the modification of the neighborhood structure, or of the goal function, such as in the *non-oblivious local search* (Khanna et al., 1994; Alimonti, 1996) where a “non-oblivious” version of the goal function is suitably constructed allowing to escape from local optima.

A completely different approach is taken by Grossi (1999), which focus the attention on the choice of a “good” initial solution as a starting point for applying the standard local search. A first and simple idea is to generate uniformly at random some different feasible solutions then taking as initial solution that one with the best value. Hence, a second step toward a deterministic approximation algorithm consist in investigating when it is possible to derandomize such initialization phase.

6.3.1 Randomized choice of initial solutions

We start by considering a very simple randomized algorithm. Let (R, g, \max) be a NPO problem, and let $N \geq 1$ be a parameter whose value will be determined in the following. Then, RandomSearch is Algorithm 6.2.

```

input  $\alpha \in \text{dom}(R)$ 
for  $i = 1, \dots, N$  do
    generate  $\beta \in \text{sol}(\alpha)$  uniformly at random
     $b_i \leftarrow \beta$ 
 $\tilde{\beta} \leftarrow \arg \max_{i \in \{1, \dots, N\}} g(\alpha, b_i)$ 
output  $\tilde{\beta}$ .

```

Algorithm 6.2: RandomSearch.

Since the focus of this section is on the approximation performance aspects of combinatorial optimization, the statement “generate $\beta \in \text{sol}(\alpha)$ uniformly at random” will be simply

⁴For a definition of such problems, see (Crescenzi and Kann, 1998).

understood as a call to a subroutine which returns, on every call, an (independent and uniformly distributed) element β of $\text{sol}(\alpha)$. A discussion of the details concerning such subroutine, in the spirit of what is studied in the other chapters of this work, is deferred to the end of this section.

A very simple analysis shows that this algorithm finds a solution whose value is “close” to the expected value of $g(\alpha, \beta)$ (when β is uniformly distributed on $\text{sol}(\alpha)$) with high confidence. More formally, define

$$E[g(\alpha, \cdot)] = \frac{1}{\#\{\beta : \beta \in \text{sol}(\alpha)\}} \sum_{\beta \in \text{sol}(\alpha)} g(\alpha, \beta);$$

then, by an application of Hoeffding’s inequality (see Appendix A), one can easily obtain the following

Proposition 6.3.1 *For every $0 < \delta < 1$ and $\varepsilon > 0$, if $N \geq 1/2\varepsilon^2 \cdot \log(1/\delta)$, the RandomSearch algorithm, on input $\alpha \in \text{dom}(R)$, gives in output β such that $g(\alpha, \beta) \geq E[g(\alpha, \cdot)] - \varepsilon \cdot \text{opt}(\alpha)$, with probability at least $1 - \delta$.*

Hence, if we restrict our attention to the class of problems for which the expectation of the goal function is a reasonable approximation of the optimum, the RandomSearch algorithm could be used as a “good” randomized approximation algorithm. We restrict again our attention to the case $\text{goal} = \max$; Grossi (1999) introduces the following class

Definition 6.3.1 *A NPO problem (R, g, \max) is RS-good iff there exists a real constant $\rho > 0$ such that, for every instance $\alpha \in \text{dom}(R)$, it holds $E[g(\alpha, \cdot)] \geq \rho \cdot \text{opt}(\alpha)$.*

In view of the analysis of the RandomSearch algorithm, we can conclude the following

Proposition 6.3.2 *If a given NPO problem (R, g, \max) is RS-good, then, for every $0 < \delta < 1$ and $\varepsilon > 0$, there exists a randomized approximation algorithm such that, on input $\alpha \in \text{dom}(R)$, returns a solution $\beta \in \text{sol}(\alpha)$ such that, with probability at least $1 - \delta$,*

$$g(\alpha, \beta) \geq (\rho - \varepsilon) \text{opt}(\alpha),$$

where $\rho > 0$ is some fixed constant independent of α .

Observe that similar definitions and results can be given for the case $\text{goal} = \min$.

Two solutions arise naturally. On one side, we can actually implement a uniform random generator for the p -relation R characterizing the NPO problem which is RS-good, hence obtaining a randomized approximation algorithm with a constant performance ratio. One of the motivation of this thesis has been to investigate which classes of relations (or combinatorial structures) admit efficient u.r.g. in order to define classes of NPO problem for which the approach taken in this section can lead to efficient randomized approximation algorithm with a constant performance ratio. Thanks to the result of Chapter 2 and 3 (in particular, from Corollary 5.1.2 and 2.2.1), one can hence conclude,

Corollary 6.3.1 *Let (R, g, goal) be a NPO problem that is RS-good and defined in terms of a p -relation that either*

- (1) *belongs to the class $\mathcal{B}(s(n), i(n), d(n))$ of p -relations recognized by extended Turing machines with simultaneous complexity bound $s(n) \cdot i(n) \cdot d(n) = O(\log n)$, or*
- (2) *is recognized by a polynomial time extended one-way nondeterministic auxiliary pushdown automaton with polynomial ambiguity.*

Then, (R, g, goal) admits a polynomial time randomized approximation algorithm with a constant performance ratio.

On the other hand, in the next section, we see that for some NPO problems, under suitable hypotheses, it is possible to deterministically obtain a solution whose measure is as good as the expected value for it. For those problems, as discussed in the following, it will be possible to have (deterministic) approximation algorithm with a constant performance ratio.

6.3.2 Derandomization: the EG class

It is well known that, for every random variable $X : \Omega \mapsto \mathbf{R}$, it always holds $E[X] \leq \sup_{\omega \in \Omega} X(\omega)$: this is the so called *internality* property of the expectation. We want to exploit this fact in relation to the analysis of the RandomSearch algorithm, considering the case in which it is possible to efficiently (and deterministically) compute a β for which $g(\alpha, \beta)$ is greater than or equal to the expectation. More formally and restricting again our attention to maximization problems, Grossi (1999), introduces the following class

Definition 6.3.2 *A NPO problem (R, g, \max) is RS-derandomizable iff there exists a polynomial time algorithm that, having in input an instance $\alpha \in \text{dom}(R)$, outputs a solution $\beta \in \text{sol}(\alpha)$ such that $g(\alpha, \beta) \geq E[g(\alpha, \cdot)]$.*

Then, by joining this property with the RS-good one, Grossi (1999) defines a class of problem for which there exists a good deterministic approximation algorithm

Definition 6.3.3 *The Expectation-Guaranteed class (EG) is the class of NPO problems that are both RS-good and RS-derandomizable.*

It is straightforward to see that, if (R, g, \max) is in EG, there exist a constant $\rho > 0$ and a polynomial time algorithm A such that, for every $\alpha \in \text{dom}(R)$,

$$g(\alpha, A(\alpha)) \geq E[g(\alpha, \cdot)] \geq \rho \cdot \text{opt}(\alpha)$$

where the first inequality follows since (R, g, \max) is RS-derandomizable and the second since it is RS-good. Hence, since similar definitions and results can be given for the case $\text{goal} = \max$, we have the following

Proposition 6.3.3 $\text{EG} \subseteq \text{APX}$.

As proved by Grossi (1999), several natural NPO problems⁵ belong to this class,

Proposition 6.3.4 *The NPO problems: “Max Sat”, “Max k -CSP”, “Max k -Cut”, “Max Bisec-tion”, “Min TSP-(1,2)” and “Max TSP-(1,2)” all belong to EG.*

In particular, as a consequence of the fact that every problem in MaxSNP can be viewed (for a suitable k) as a “Max k -CSP” problem (Khanna et al., 1994), and by the previously recalled proposition, one can conclude that

Proposition 6.3.5 $\text{MaxSNP} \subseteq \text{EG}$.

To conclude this subsection, we recall that Grossi (1999) has also proven the following

Proposition 6.3.6 *The inclusions $\text{MaxSNP} \subsetneq \text{EG} \subsetneq \text{APX}$ are strict. Moreover, $\overline{\text{EG}} = \text{APX}$, where the closure is taken with respect to the AP-reduction.*

6.3.3 The method of conditional expectation

In this subsection we discuss the *method of conditional expectation* (Erdős and Spencer, 1974): a particular technique which allows to show that some NPO problem, under suitable hypotheses, is RS-derandomizable.

Let (R, g, \max) be a NPO problem over some alphabet Σ , where R is a p -relation. For every instance $\alpha \in \text{dom}(R)$ and $1 \leq k \leq p(|\alpha|)$, define the subset of feasible solutions $\text{sol}(\alpha; b_1, \dots, b_k) = \{\beta_1 \dots \beta_{p(|\alpha|)} \in \text{sol}(\alpha) : \beta_j = b_j \text{ for } 1 \leq j \leq k\}$ having prefix $b_1, \dots, b_k \in \Sigma$. Then, the expectation of $g(\alpha, \beta)$ when β is uniformly distributed on $\text{sol}(\alpha)$, conditionally to β having prefix $b_1, \dots, b_k \in \Sigma$, can be defined as

$$\mathbb{E}[g(\alpha, \cdot) \mid b_1, \dots, b_k] = \frac{1}{\#\{\beta : \beta \in \text{sol}(\alpha; b_1, \dots, b_k)\}} \sum_{\beta \in \text{sol}(\alpha; b_1, \dots, b_k)} g(\alpha, \beta).$$

By computing such conditional expectation on prefixes of increasing length, one can incrementally compute the β for which $g(\alpha, \beta)$ is greater than or equal to $\mathbb{E}[g(\alpha, \cdot)]$, selecting for each “position” of the prefix the best possible value. More formally, consider Algorithm 6.3.

By the properties of conditional expectation⁶, one can verify that, for every $b_1, \dots, b_k \in \Sigma$,

$$\mathbb{E}[g(\alpha, \cdot) \mid b_1, \dots, b_k] \leq \max_{b \in \Sigma} \mathbb{E}[g(\alpha, \cdot) \mid b_1, \dots, b_k, b];$$

hence, a simple analysis shows that, if β is the output of the previous algorithm on input α , then $g(\alpha, \beta) \geq \mathbb{E}[g(\alpha, \cdot)]$.

⁵For a definition of such problems, see (Crescenzi and Kann, 1998).

⁶for a detailed discussion of the probabilistic aspects of such method, see Appendix A.2.3.

```

input  $\alpha \in \text{dom}(R)$ 
for  $k = 1, \dots, p(|\alpha|)$  do
     $a_k \leftarrow \arg \max_{b \in \Sigma} E[g(\alpha, \cdot) \mid a_1, \dots, a_{k-1}, b]$ 
 $\beta \leftarrow a_1 \dots a_{p(|\alpha|)}$ 
output  $\beta$ .

```

Algorithm 6.3: Conditional expectation.

Let now $T(n, k)$ be the maximum time required, by a suitable algorithm, to compute $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$ for all $\alpha \in \text{dom}(R)$ such that $|\alpha| = n$. Then, since the $\arg \max$ is computed over the alphabet Σ that is finite by definition, the computation time of the above algorithm is $O(\sum_{k=1}^{p(n)} T(n, k))$. Hence, we can conclude with the following sufficient condition for establishing whether a NPO problem is RS-derandomizable (Grossi, 1999)

Proposition 6.3.7 *If a NPO problem (R, g, goal) is such that $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$ is computable in time polynomial in $|\alpha|$ and k , then (R, g, goal) is RS-derandomizable.*

When all solutions are feasible

Now we investigate, in the particular case when all solutions are feasible, a sufficient condition for establishing that $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$ is computable in polynomial time. Assume for simplicity⁷ $\Sigma = \{0, 1\}$ and, as already stated, consider a NPO problem (R, g, goal) such that $\text{sol}(\alpha) = \Sigma^{p(|\alpha|)}$, for all $\alpha \in \text{dom}(R)$ (where p is the polynomial for which R is a p -relation). Fix now some $\alpha \in \text{dom}(R)$ and let $n = p(|\alpha|)$; the goal function g can be written as

$$g(\alpha, \beta_1 \dots \beta_n) = \sum_{(\lambda_1, \dots, \lambda_n) \in \{0, 1\}^n} \hat{g}(\alpha, \lambda_1, \dots, \lambda_n) \beta_1^{\lambda_1} \dots \beta_n^{\lambda_n} = G_\alpha(\beta_1, \dots, \beta_n) \quad (6.1)$$

where $G_\alpha(\beta_1, \dots, \beta_n)$ is a n -th variate polynomial over $\{0, 1\}^n$ whose variables have degree at most one (and $\hat{g}(\alpha, \lambda_1, \dots, \lambda_n)$ are considered as coefficient in \mathbf{N}).

In the next section a more detailed discussion on polynomials and on their relation with pseudo-boolean functions will be carried on. Here, instead, we only want to sketch how this simple idea can lead to compute $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$. First of all, recall that the expectation is a linear functional, such a relevant property that Erdős ironically used to call it “Adam’s theorem”! Moreover, since the expectation is taken with respect to the uniform distribution on $\beta \in \Sigma^n$, we can actually consider the β_i as “independent”. From this very simple observations, it follows that

$$\begin{aligned}
 E[g(\alpha, \cdot) \mid b_1, \dots, b_k] &= E[G_\alpha(\beta_1, \dots, \beta_n) \mid b_1, \dots, b_k] \\
 &= \sum_{(\lambda_1, \dots, \lambda_n) \in \{0, 1\}^n} g(\alpha, \lambda_1 \dots \lambda_n) E[\beta_1^{\lambda_1} \dots \beta_n^{\lambda_n} \mid b_1, \dots, b_k] \\
 &= \sum_{(\lambda_1, \dots, \lambda_n) \in \{0, 1\}^n} g(\alpha, \lambda_1 \dots \lambda_n) b_1^{\lambda_1} \dots b_k^{\lambda_k} \prod_{j=k+1}^n E[\beta_j^{\lambda_j}] \\
 &= \sum_{(\lambda_1, \dots, \lambda_n) \in \{0, 1\}^n} g(\alpha, \lambda_1 \dots \lambda_n) b_1^{\lambda_1} \dots b_k^{\lambda_k} \left(\frac{1}{2}\right)^{n-k} \\
 &= G_\alpha(b_1, \dots, b_k, 1/2, \dots, 1/2).
 \end{aligned}$$

⁷Observe that this is not a limitation, since one can always encode over $\{0, 1\}$ every finite alphabet Σ .

We can summarize this discussion by the following (Grossi, 1999)

Proposition 6.3.8 *Let (R, g, goal) be a NPO problem on $\Sigma = \{0, 1\}$ such that $\text{sol}(\alpha) = \Sigma^{p(|\alpha|)}$ for every $\alpha \in \text{dom}(R)$ and let G_α be defined as in equation (6.1). If there exists an algorithm that, for every $\alpha \in \text{dom}(R)$, evaluates G_α on $\{0, 1, 1/2\}^{p(|\alpha|)}$ in time polynomial in $|\alpha|$, then (R, g, goal) is RS-derandomizable.*

6.3.4 Derandomization and the EG class

As suggested by Grossi (1999), if a NPO problem is in EG, then by combining derandomization and local search, one can obtain an efficient approximation algorithm. As an example, assuming the condition of Proposition 6.3.8, for a given neighborhood structure $\beta \mapsto S_\beta$ (see Section 6.2.1), we can have Algorithm 6.4.

input $\alpha \in \text{dom}(R)$

Step 1
for $k = 1, \dots, p(|\alpha|)$ **do**
 $a_k \leftarrow \arg \max_{b \in \{0, 1\}} G_\alpha(a_1, \dots, a_{k-1}, b, 1/2, \dots, 1/2)$
 $\beta \leftarrow a_1 \dots a_{p(|\alpha|)}$

Step 2
repeat
 pick $\beta' \in S_\beta$
 if $g(\alpha, \beta') > g(\alpha, \beta)$ **then** $\beta \leftarrow \beta'$
until $f(\beta') \leq f(\beta)$ for all $\beta' \in S_\beta$

output β .

Algorithm 6.4: An approximation algorithm for EG problems.

Following this idea, but in a more general fashion, Grossi (1999) proved the existence of a “universal” schema of algorithm for all the problems in EG; more formally

Proposition 6.3.9 *For every NPO problem (R, g, goal) in EG, (with respect to a suitable h -bounded neighborhood) there exists a ρ -approximation algorithm for (R, g, goal) where*

$$\rho = \sup_{\alpha \in \text{dom}(R)} \frac{\text{opt}(\alpha)}{E[g(\alpha, \cdot)]}.$$

This schema of algorithm is based on the computation of $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$, hence, in order to obtain polynomial time algorithms following such a schema, $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$ should be computable in polynomial time.

It is then clear that, in order to apply the derandomized initialization phase to local search algorithm as discussed in this section, it is of fundamental importance to determine whether $E[g(\alpha, \cdot) \mid b_1, \dots, b_k]$, or $G_\alpha(a_1, \dots, a_{k-1}, b, 1/2, \dots, 1/2)$ (as defined by equation (6.1)) are computable in polynomial time. This is exactly the aim of the next section.

6.4 Derandomization and Arithmetic Circuits

With respect to the results about derandomization discussed in Subsection 6.3.3, here we investigate, for a given NPO problem (R, g, goal) over $\Sigma = \{0, 1\}$, whenever it is possible to design algorithms to efficiently evaluate G_α having as input some specification of g (and α). To this end, we use the notion of *arithmetic circuit* as a way to specify pseudo-boolean functions (such as G_α and g). The semantic of arithmetic circuits, and some relevant properties about them, are expressed using polynomials, which are here formally defined.

6.4.1 Polynomials and arithmetic circuits

Polynomials

We begin by recalling some elementary facts about polynomials; for a thorough survey see, for example, (Lang, 1994). Given a ring R we denote with $R[x_1, \dots, x_n]$ the (commutative) R -algebra free over $\{x_1, \dots, x_n\}$ of *polynomials* in the *indeterminates* x_1, \dots, x_n . The *degree* $\deg(p)$ of a polynomial $p \in R[x_1, \dots, x_n]$ is the maximum degree of its monomials which, in turn, is simply the sum of the exponents of the variables appearing in the monomial. Given a R -algebra \mathcal{A} and $\mathbf{a} \in \mathcal{A}^n$ we denote with $p(\mathbf{a}) \in \mathcal{A}$ the *evaluation* of p in \mathbf{a} obtained by “replacing” each x_i with a_i in p ; if $\mathcal{A} = R[y_1, \dots, y_m]$ and $q_1, \dots, q_n \in R[y_1, \dots, y_m]$ we denote with $p[q_1, \dots, q_n] \in R[y_1, \dots, y_m]$ the *substitution* of x_1, \dots, x_n with q_1, \dots, q_n . We now give some definitions needed in the following:

Definition 6.4.1 *Given a polynomial $p \in R[x_1, \dots, x_n]$ and a monomial m we denote by $[m](p)$ the coefficient of m in p (0 if m is not in p) and with $\text{msf}(p)$ the polynomial obtained by substituting every x_i^k in p with x_i , for each $1 \leq i \leq n$ and $k > 0$. We denote by $R_{\text{msf}}[x_1, \dots, x_n]$ the set of polynomials in $R[x_1, \dots, x_n]$ whose monomials are square-free.*

It is well known that, in some cases, a polynomial can be uniquely determined given a suitable number of its evaluations; in particular, we prove the following:

Lemma 6.4.1 *Any $p \in R_{\text{msf}}[x_1, \dots, x_n]$ is uniquely determined by its evaluations over $\{0, 1\}^n$.*

Proof . Let us introduce some notation to allow a more concise proof. For $1 \leq i \leq n$, define $\mathbf{e}^i = (e_1^i, \dots, e_n^i) \in \{0, 1\}^n$ such that $e_j^i = 1$ iff $i = j$; for every $I \subseteq \{1, \dots, n\}$, let $\mathbf{e}^I = \sum_{i \in I} \mathbf{e}^i \in \{0, 1\}^n$ and define $\mathbf{e}^\emptyset = (0, \dots, 0)$. Finally, denote by \mathbf{x}^I the monomial $\prod_{i \in I} x_i$, where \mathbf{x}^\emptyset denotes the unit of $R[x_1, \dots, x_n]$. It is straightforward to verify that $p(\mathbf{e}^I) = \sum_{J \subseteq I} [\mathbf{x}^J]p = [\mathbf{x}^I]p + \sum_{J \subsetneq I} [\mathbf{x}^J]p$, that is $[\mathbf{x}^I]p = p(\mathbf{e}^I) - \sum_{J \subsetneq I} [\mathbf{x}^J]p$. Hence, by induction on the cardinality of I , is possible to verify that the coefficient of every monomial \mathbf{x}^I of p is uniquely determined by evaluating $p(\mathbf{e}^J)$ for $J \subseteq I$. ■

Arithmetic circuits

We observe that various definitions of arithmetic circuits are present in literature (see, for instance, the works of von zur Gathen and Seroussi (1991) or Valiant, Skyum, Berkowitz, and Rackoff (1983)) and that the differences between those definitions mainly concern the in-degree or out-degree.

Definition 6.4.2 An arithmetic circuit over the ring R with inputs x_1, \dots, x_n and constants from $S \subseteq R$ is a node-labelled direct acyclic graph such that: nodes with in-degree 0 are labeled with elements of $S \cup \{x_1, \dots, x_n\}$, nodes with positive in-degree are labeled with elements of $\{+, \times\}$ and there is a distinguished node, the only one with out-degree 0, called output node. The set of all arithmetic circuits over the ring R with inputs x_1, \dots, x_n and constants from S will be denoted by $\mathcal{C}_R^S[x_1, \dots, x_n]$. The size $C(c)$ of a circuit c is the number of nodes and its depth $D(c)$ is the length of the longest path ending in the output node of c .

With every node v of an arithmetic circuit over R we recursively associate a polynomial $p \in R[x_1, \dots, x_n]$ according to its label ℓ :

- if $\ell \in S \cup \{x_1, \dots, x_n\}$, then $p = \ell$;
- if $\ell \in \{+, \times\}$, then $p = p_1 \ell p_2 \dots \ell p_k$ where p_1, \dots, p_k are the polynomials associated to the predecessors of v .

We also denote with c the polynomial associated to the output node of a circuit c , so by degree (maximum degree) of a circuit c we mean the degree (respectively, maximum degree) of the polynomial c .

Since the main statements of this section are concerned with circuits of size polynomial in the number of inputs, it can be easily verified that even when restricting the in-degree and out-degree to 2 in the previous definition, none of the presented results would be substantially altered.

Circuits representing polynomials

In the following we will restrict our attention to the class of circuits $\mathcal{C}_{\mathbf{Z}}^{\{-1,0,1\}}[x_1, \dots, x_n]$ which we will simply denote with \mathcal{C}_n , using $\mathcal{C}_n^{\text{msf}}$ to denote the subset of \mathcal{C}_n whose circuits have their associated polynomial in $\mathbf{Z}_{\text{msf}}[x_1, \dots, x_n]$. These circuits are interesting since they can be effectively encoded over a finite alphabet (in size $C(c)^{O(1)}$ if the alphabet is at least binary) and then some computations over them can be performed by means of algorithms. We give now an example of such a computation:

Lemma 6.4.2 Given inputs $c \in \mathcal{C}_n^{\text{msf}}$ and a monomial m , $[m]c \in \mathbf{Z}$ can be computed in $O(C(c)^2 d \log d)$ time, where $d = \min\{\deg(m), n\}$.

Proof . If m is not square-free, then $[m]c = 0$. Otherwise $d(m) \leq n$ and m can be written as $x_1^{\lambda_1} \dots x_n^{\lambda_n}$ with $\lambda_1, \dots, \lambda_n \in \{0, 1\}$: define $p = c[\lambda_1 z, \dots, \lambda_n z] \in \mathbf{Z}[z]$. It is straightforward to observe that $d(p) = d$ and $[z^d]p = [m]c$. The coefficients of p are in \mathbf{Z} and can be computed using the (encoding of) circuit c : the computation requires, for each of the $C(c)$ nodes of c , the sum or product of the polynomials associated to its predecessors that are at most $C(c)$. Since all the involved polynomials are in $\mathbf{Z}[z]$, their sum or product requires at most $d \log d$ steps; given p it is then immediate to get the coefficient of z^d , hence the coefficient of m in c . ■

In the following, as we have done in this lemma with a small abuse of notation, we will identify a circuit and its encoding. We are interested in circuits of \mathcal{C}_n “representing” polynomials; more precisely:

Definition 6.4.3 A circuit $c \in \mathcal{C}_R^S[x_1, \dots, x_n]$ represents $p \in R[x_1, \dots, x_n]$ over some subset A of a R -algebra whenever $c(\mathbf{a}) = p(\mathbf{a})$ for all $\mathbf{a} \in A^n$. A circuit $c \in \mathcal{C}_R^S[x_1, \dots, x_n]$ exactly represents $p \in R[x_1, \dots, x_n]$ whenever $c = p$ (as polynomials).

Some interesting facts are known about the representation of polynomials by means of arithmetic circuits in \mathcal{C}_n and about the relationships between circuits representing the same polynomial:

Proposition 6.4.1 For every $p \in \mathbf{Z}[x_1, \dots, x_n]$ there exists $c \in \mathcal{C}_n$ exactly representing p (and thus representing $\text{msf}(p)$ over $\{0, 1\}$) and $c' \in \mathcal{C}_n^{\text{msf}}$ exactly representing $\text{msf}(p)$.

It is clear that each polynomial p (or $\text{msf}(p)$) immediately “suggests” a circuit; moreover, noting that $x^k = x$ when $x \in \{0, 1\}$ (for $k \geq 1$), it is also evident that if c exactly represents p , then c also represents $\text{msf}(p)$ over $\{0, 1\}$. Since the number of monomials in n indeterminates of maximum degree at most 1 is 2^n , the size of the circuits naïvely obtained from $\text{msf}(p)$ can, in principle, be exponential in n . By a simple application of previous observations and of Lemma 6.4.1 it is simple to notice that:

Proposition 6.4.2 If $c \in \mathcal{C}_n^{\text{msf}}$ represents $p \in \mathbf{Z}[x_1, \dots, x_n]$ over $\{0, 1\}$, then c exactly represents $\text{msf}(p)$.

6.4.2 Back to our problem

Recall that our aim was to investigate when it is possible to design algorithms to efficiently evaluate G_α having as input some specification of g (and α), for some NPO problem (R, g, goal) where R is a p -relation.

We have chosen arithmetic circuits as a way to specify pseudo-boolean functions: by this we mean that g is specified, for every $\alpha \in \text{dom}(R)$, by a family of circuits $\{c_n\}_{n>0}$ with $c_n \in \mathcal{C}_n$ such that, for every $\beta \in \Sigma^m$ with $m = p(|\alpha|)$, one has that $g(\alpha, \beta) = c_m(\beta_1, \dots, \beta_m)$. It is also clear from the definition of G_α that this is a polynomial in $\mathbf{Z}_{\text{msf}}[x_1, \dots, x_n]$, hence it should be specified by a family of circuits $\{c'_n\}_{n>0}$ with $c'_n \in \mathcal{C}_n^{\text{msf}}$ such that $G_\alpha(\beta_1, \dots, \beta_m) = c'_m(\beta_1, \dots, \beta_m)$.

Hence, we are interested in the relationship between circuits in \mathcal{C}_n and $\mathcal{C}_n^{\text{msf}}$; in particular we would like to find efficient (i.e., polynomial-time) algorithms to transform circuits from one class to the other somehow preserving their evaluation. Some similar results are known: for instance Valiant et al. (1983) have proven

Proposition 6.4.3 For every $c \in \mathcal{C}_n$ a circuit $c' \in \mathcal{C}_n$ can be computed in time $C(c)^{O(1)}$ such that c' exactly represents c and has size $O(C(c)^3)$ and depth $O(\log C(c) \log \deg(c))$.

We now introduce a polynomial that is central to the rest of this note:

Definition 6.4.4 Let $A = (a_{ij})$ be a $n \times n$ matrix with values in $\{0, 1\}$ and consider the function $f_A(\mathbf{x}) = \prod_{i=1}^n \sum_{j=1}^n a_{ij} x_j$; let $p_A \in \mathbf{Z}[x_1, \dots, x_n]$ be the polynomial whose evaluation coincides with f_A over \mathbf{Z}^n .

The polynomial p_A has some interesting properties investigated in the next:

Lemma 6.4.3 $\text{perm}(A) = [x_1 \cdots x_n] \text{msf}(p_A)$ and, for every $s \in \mathbf{N}$ such that $\text{perm}(A) < 2^s$, $\text{perm}(A) = 2^s \{2^{s(n-1)} \text{msf}(p_A)(2^{-s}, \dots, 2^{-s})\}$ (where $\{x\}$ is the fractional part of $x \in \mathbf{R}$).

Proof . Since $p_A = \sum a_{1i_1} \cdots a_{ni_n} x_{i_1} \cdots x_{i_n}$, where the summation extends over all the n -tuples $(i_1, \dots, i_n) \in \{1, \dots, n\}^n$, it is easy to conclude that $[x_1 \cdots x_n] p_A = \text{perm}(A)$, given that the monomial $x_1 \cdots x_n$ “selects” from the summation only those terms corresponding to tuples that are a permutation of $\{1, \dots, n\}$. Then, observing that the only monomial of degree n in $\text{msf}(p_A)$ is $x_1 \cdots x_n$, it is clear that $[x_1 \cdots x_n] p_A = [x_1 \cdots x_n] \text{msf}(p_A)$. Moreover, letting $q_A = \text{msf}(p_A)[z, \dots, z] \in \mathbf{Z}[z]$, it is straightforward to verify that $q_A(z) = z^n \text{perm}(A) + q'_A(z)$ with $q'_A \in \mathbf{Z}[z]$ such that $d(q'_A) \leq n-1$. Hence, for every s such that $\text{perm}(A)/2^s < 1$, we have $2^{s(n-1)} q_A(2^{-s}) = \text{perm}(A)/2^s + 2^{s(n-1)} q'_A(2^{-s})$ thus, since $2^{s(n-1)} q'_A(2^{-s}) \in \mathbf{N}$, $\text{perm}(A)$ can be obtained from the fractional part of $2^{s(n-1)} q_A(2^{-s})$. ■

We begin with an introductory result about algorithms transforming circuits $c \in \mathcal{C}_n$ to $c' \in \mathcal{C}_n^{\text{msf}}$ such that c' represents c over $\{0, 1\}$. It is straightforward to observe that:

Proposition 6.4.4 *There exists $c_A \in \mathcal{C}_n$ exactly representing p_A of size $n^{O(1)}$ and constant depth.*

The circuit c_A , which clearly represents p_A over $\{0, 1\}$, can be obtained from f_A in a straightforward way. Nonetheless we will show that:

Theorem 6.4.1 *If there exists $c \in \mathcal{C}_n^{\text{msf}}$ of size $n^{O(1)}$ representing p_A over $\{0, 1\}$, then $\text{perm}(A)$ can be computed in time $n^{O(1)}$ having c as input.*

Proof . Since c exactly represents $\text{msf}(p_A)$ by Proposition 6.4.2, then $[m]c = [m] \text{msf}(p_A)$ for every monomial m ; hence, if $C(c) = n^{O(1)}$, by lemmas 6.4.2 and 6.4.3, $\text{perm}(A)$ can be computed, having c as input, in $O(C(c)^2 n \log n) = n^{O(1)}$ time, where $n = \deg(x_1 \cdots x_n)$. ■

Given that the computation of the permanent is a $\sharp\text{P}$ -complete⁸ problem (Valiant, 1979), we can conclude that:

Proposition 6.4.5 *The problem of mapping a circuit $c \in \mathcal{C}_n$ to a circuit $c' \in \mathcal{C}_n^{\text{msf}}$ of size $n^{O(1)}$ representing c over $\{0, 1\}$, is $\sharp\text{P}$ -hard.*

We now give a stronger result by weakening the hypotheses of Theorem 6.4.1; in the transformation from $c \in \mathcal{C}_n$ to $c' \in \mathcal{C}_n^{\text{msf}}$, it will now be required that c' represents $\text{msf}(c)$ only over $\{1/2\}$.

We need introduce a new polynomial, depending on p_A . Fix some $s \in \mathbf{N}$ and consider the substitution replacing x_i with $q_i = x_{i1} \cdots x_{is}$ for $1 \leq i \leq n$; define the function $\tilde{f}_{A,s}(x_{11}, \dots, x_{ns}) = f_A(q_1(x_{11} \cdots x_{1s}), \dots, q_n(x_{n1} \cdots x_{ns}))$ and let $\tilde{p}_{A,s} \in \mathbf{Z}[x_{11}, \dots, x_{ns}]$ be the polynomial whose evaluation coincides with \tilde{f}_A over \mathbf{Z}^{ns} . It is straightforward to note that $p_A[q_1, \dots, q_n] = \tilde{p}_{A,s}$ and that $\text{msf}(\tilde{p}_{A,s}) = \text{msf}(p_A[q_1, \dots, q_n])$, since q_i and q_j have no common indeterminates if $i \neq j$. In analogy with the case of p_A , an efficient representation for the polynomial $\tilde{p}_{A,s}$ can be obviously deduced from its definition:

Proposition 6.4.6 *There exists $\tilde{c}_A \in \mathcal{C}_{ns}$ exactly representing $\tilde{p}_{A,s}$ of size $(ns)^{O(1)}$ and constant depth.*

⁸here, as in (Garey and Johnson, 1979), completeness and hardness for $\sharp\text{P}$ are given with respect to Turing reductions.

Nonetheless, having a small circuit representing $\text{msf}(\tilde{p}_{A,s})$ even on only one value can lead to the computation of $\text{perm}(A)$:

Theorem 6.4.2 *If there exists $c \in \mathcal{C}_{n^3}^{\text{msf}}$ of size $n^{O(1)}$ representing $\text{msf}(\tilde{p}_{A,n^2})$ over $\{1/2\}$, then $\text{perm}(A)$ can be computed in $n^{O(1)}$ time having c as input.*

Proof . By the definition of $\tilde{p}_{A,s}$ it should be clear that $\text{msf}(p_A)(2^{-s}, \dots, 2^{-s}) = \text{msf}(\tilde{p}_{A,s})(1/2, \dots, 1/2)$ and, by the hypotheses of the theorem, $\text{msf}(\tilde{p}_{A,s})(1/2, \dots, 1/2) = c(1/2, \dots, 1/2)$ and it can be computed, having c as input, in $n^{O(1)}$ steps. Since $a_{ij} \in \{0, 1\}$, $\text{perm}(A) < 2^{n^2}$ so, by Lemma 6.4.3, $\text{perm}(A)$ can be computed taking the fractional part of $2^{n^2(n-1)}c(1/2, \dots, 1/2)$ and then multiplying by 2^{n^2} which can be done in overall time $n^{O(1)}$. ■

We are finally able to conclude with the main statement of this chapter:

Proposition 6.4.7 *The problem of mapping a circuit $c \in \mathcal{C}_n$, exactly representing $p \in \mathbf{Z}[x_1, \dots, x_n]$, to a circuit $c' \in \mathcal{C}_n^{\text{msf}}$ of size $n^{O(1)}$ representing $\text{msf}(p)$ over $\{1/2\}$ is $\sharp\text{P}$ -hard.*

6.5 Conclusions

In this chapter we have discussed an application of uniform random generation to combinatorial optimization. As we have seen, for NPO problems which are RS-good, the existence of a polynomial time u.r.g. leads to a polynomial time randomized approximation algorithm with a constant performance ratio. The results of this thesis about the existence of polynomial time u.r.g. for several classes of p -relations and languages can be applied to obtain such approximation algorithms.

On the other hand, for NPO problems that are also RS-derandomizable, i.e., for EG problems, there also exists the possibility to obtain polynomial time deterministic approximation algorithm with a constant performance ratio. In particular, whenever the method of conditional expectation can be efficiently applied, a “universal” polynomial time approximation algorithm can be designed for every problem in EG.

Nonetheless, in the last section, we have shown that given a NPO problem (R, g, goal) over $\Sigma = \{0, 1\}$ where g is a pseudo-boolean function specified by means of arithmetic circuit of small size, it is in general a $\sharp\text{P}$ -hard problem to obtain a corresponding circuit to evaluate G_α (as defined by equation (6.1) of Section 6.3.3) over $\{0, 1\}$, or even over $1/2$. This negative result shows a limit to the derandomization approach to combinatorial optimization and gives some evidence that the abovementioned “universal” approach for solving problems in EG cannot efficiently solve every such problem (unless $\text{FP} = \sharp\text{P}$).

CHAPTER 7

CONCLUSIONS AND OPEN PROBLEMS

...
*so that there is no sun and no unveiling
and no host
only I and then the sheet
and bulk dead*

Samuel Beckett, *Poems in English*

The aim of this thesis has been to investigate, from the structural complexity point of view, the relationships between (exact and approximate) *counting*, *ranking* and *uniform random generation* and to determine some new classes of combinatorial structures admitting efficient approximate counting and uniform random generation.

We have chosen the PrRAM as a *formal model of computation* for two main reasons. On one side, the PrRAM model is polynomially related to the classical probabilistic Turing machine model so that it can be considered, in the usual structural complexity framework, as a *feasible* model of randomized computation. On the other side, the PrRAM showed to be of sufficient *high level* to let us design the algorithms of this thesis avoiding the boring specifications that usually arise when dealing directly with the Turing machine model.

In describing formally the various problems we dealt with, we have used the notion of *p-relation*, *combinatorial structure* and *formal language* in a somehow decreasing order of generality to suggest the broad applicability of the presented results avoiding at the same time unnecessary technicalities in some of the given definitions and proofs.

To solve the problem of random generation, the first simple idea is related to the ranking of formal languages: if we are able to determine a string in a formal language given its position (i.e., to solve the unranking problem for that language), the problem of random generation of

strings in such a language can be reduced to the random generation of (bounded) integers. We have exploited this idea in the general setting of p -relations, where the unranking problem is efficiently solvable whenever the rank can be efficiently computed. We have then extended some results about the ranking of formal languages based on particular acceptor devices to the case of p -relations thus obtaining two new classes of p -relations admitting an efficient solution to the uniform random generation problem.

With regard to this approach, it remains as an open problem to investigate other classes of p -relations admitting efficient ranking. For instance, we are now interested in solving the ranking problem for languages accepted by automata endowed with both stack and queues, as introduced by Breveglieri, Cherubini, and Crespi-Reghizzi (1991); Breveglieri, Cherubini, Citrini, and Crespi-Reghizzi (1996), used to describe formally, for instance, various operating system scheduling strategies.

We have also noted that there are cases in which the problem of uniform random generation is efficiently solvable when the ranking problem is computationally intractable. This happens, for instance, by taking the union of some unambiguous context-free languages. In such cases, a very different approach still yields to an efficient solution of the random generation and approximate counting problems. This is for instance the case when the combinatorial structure whose elements we want to generate uniformly at random can be, in some precise sense, described by means of another structure which in turn admits efficient uniform random generation. We have applied such general framework of (ambiguous) descriptions to some classes of formal languages, in particular to the case of rational trace languages, languages accepted by polynomial time nondeterministic auxiliary pushdown automata (and hence context-free languages). In each case, we have solved in an efficient way the problem of uniform random generation and approximate counting whenever the ambiguity of the given language is bounded (finite, or polynomial).

It remains as an open problem to investigate further examples of applications of the (ambiguous) descriptions framework. We are in particular interested to approximate solutions for difficult counting problems obtained via the use of (ambiguous) descriptions. Some examples of the kind of problems we are interested in can be given, for instance, by the coefficients of algebraic power series arising in the study of formal languages Choffrut and Goldwurm (1995).

Finally, we have applied our results about random generation to an heuristic for improving local search algorithms used in combinatorial optimization. This application is in particular motivated from the fact that we have proved that the derandomization of such an heuristic can, in some case, be a $\sharp P$ -hard problem.

With regard to the latter issue, it remains as an open problem to relate the formal results obtained here in terms of acceptor devices to various examples of natural combinatorial optimization problems as presented, for instance, in Crescenzi and Kann (1998).

APPENDIX A

TECHNICALITIES

It was as if these depths constantly bridged over by a structure that was firm enough in spite of its lightness and of its occasional oscillation in the somewhat vertiginous air, invited on occasion, in the interest of their nerves, a dropping of the plummet and a measurement of the abyss.

Henry James, *The Beast in the Jungle*

The aim of this appendix is to separately discuss some technical and probabilistic aspects of this thesis in order not to crowd with details its main part.

A.1 Some Lemmas

In this section we give the proof of some technical results that are needed in the thesis. We start by an inequality we have often used in proving several upper bounds:

Lemma A.1.1 *For every $\alpha > 0$ and $\beta > 0$, there exists a constant $\kappa > 0$ such that, for every $0 < \varepsilon < 1/\beta$ and $\delta > 0$, it holds that*

$$\alpha(1 - \beta\varepsilon)^{\frac{\kappa}{\varepsilon} \max\{1, \log(1/\delta)\}} < \delta.$$

Proof . By taking the logs, the above inequality becomes $\kappa \max\{1, \log(1/\delta)\}/\varepsilon \log(1 - \beta\varepsilon) < \log(\delta/\alpha)$. Since $\log(1 - x) \leq -x$, for $x \leq 1$, the inequality holds whenever

$$-\kappa\beta \max\{1, \log(1/\delta)\} < \log(\delta/\alpha).$$

Therefore, if $\kappa > (1 + \max\{0, \log \alpha\})/\beta > 0$, we have

$$\begin{aligned} \kappa &> \frac{1}{\beta} \left(\frac{\log(1/\delta)}{\max\{1, \log(1/\delta)\}} + \frac{\max\{0, \log \alpha\}}{\max\{1, \log(1/\delta)\}} \right) \geq \\ &\geq \frac{1}{\beta} \frac{\log(1/\delta) + \log \alpha}{\max\{1, \log(1/\delta)\}} = -\frac{\log(\delta/\alpha)}{\beta \max\{1, \log(1/\delta)\}}. \blacksquare \end{aligned}$$

Now we consider two algorithms that, respectively, compute the least common multiple (lcm) of the first n integers and the integer part of the logarithm of n :

Lemma A.1.2 *The least common multiple (lcm) of $\{1, \dots, n\}$ is an integer of $O(n)$ bits and it can be computed in $O(n^2)$ time by a RAM under logarithmic cost criterion.*

Proof . First, we recall that $\text{lcm}\{1, \dots, n\} \leq n^{\pi(n)}$, where $\pi(n)$ is the n -th prime (Szepietowski, 1994, Lemma 4.1.2.). As a consequence, $\text{lcm}\{1, \dots, n\}$ has $O(n)$ bits since it is well known that $\pi(n) \sim n/\log n$. Moreover, a naïve iterative procedure, based on Euclid's algorithm for computing the g.c.d. can be designed to compute $\text{lcm}\{1, \dots, n\}$ in $O(n^2)$ time. ■

Lemma A.1.3 *The number $\lceil \log N \rceil$ of bits required to represent $\{1, \dots, N\}$ can be computed in $O(\log N \log \log N)$ by a RAM under logarithmic cost criterion.*

Proof . We compute $\lceil \log N \rceil$ by the call $\text{Size}(N - 1)$, where “Size” is the recursive procedure given by Algorithm A.1.

```

procedure Size( $n$ )
if  $n \leq 1$  then return 1
else
     $h \leftarrow 1, h_0 \leftarrow h$ 
     $k \leftarrow 2, k_0 \leftarrow k$ 
    while  $k \leq n$  do
         $h_0 \leftarrow h, h \leftarrow h + h$ 
         $k_0 \leftarrow k, k \leftarrow k \cdot k$ 
    return  $h_0 + \text{Size}(n/k_0)$ .
```

Algorithm A.1: Computing the bit size of an integer.

A simple analysis shows that the time complexity of $\text{Size}(n)$ is given by $O(\sum_{i=1}^k i2^i) = O(k2^k)$, where $k = \log \log n$. ■

A.2 Probabilistic Detour

This section is intended to give a separate discussion of the probabilistic aspects involved in the thesis. In this way, we hope to give a more precise and detailed evidence of some of the properties used in order to prove the correctness of algorithms we have presented in the previous chapters.

A.2.1 Hoeffding's inequality

Hoeffding's inequality (Hoeffding, 1963) is the main tool used in this work to prove concentration results, i.e., to show that the sum of certain random variables has, with high probability, a value very near to its expectation. This inequality is based on the so called *Chernoff's bounding technique* (Chernoff, 1952) that uses an exponential version of *Markov's inequality*. Here, we

want to give a concise proof of such an inequality since is based on a very simple and intuitive idea we would like to elucidate.

Let $\langle \Omega, \mathcal{F}, P \rangle$ be a probability space, and consider a random variable X on it, i.e., a measurable function $X : \Omega \rightarrow \mathbf{R}$. The *expectation* of X , denoted by $E[X]$, is defined as

$$E[X] = \int_{\Omega} X dP$$

where the integral is understood in the Lebesgue sense (for a detailed discussion of these details, see, for instance (Billingsley, 1979; Feller, 1968)). If, for every $\omega \in \Omega$, it holds that $X(\omega) > 0$, then, for every $t > 0$,

$$\int_{\Omega} X dP \geq \int_{X \geq t} t dP + \int_{X \leq t} X dP \geq tP\{X \geq t\} + 0.$$

Hence, we have proven *Markov's inequality*:

$$P\{X \geq t\} \leq \frac{E[X]}{t}.$$

Now, by taking the exponential, we can apply this inequality for every random variable Y , even those assuming negative values: for every $\alpha > 0$, we have

$$P\{Y \geq t\} = P\{e^{\alpha Y} \geq e^{\alpha t}\} \leq \frac{E[e^{\alpha Y}]}{e^{\alpha t}};$$

Chernoff's bounding idea consists in choosing α such that the right hand side of the previous inequality is minimized. We can apply the inequality to a sum of independent random variables X_1, \dots, X_n :

$$\begin{aligned} P\left\{\left(\sum_{i=1}^n X_i - E\left[\sum_{i=1}^n X_i\right]\right) \geq t\right\} &\leq e^{-\alpha t} E\left[\exp\left(\alpha \sum_{i=1}^n (X_i - E[X_i])\right)\right] \\ &= e^{-\alpha t} \prod_{i=1}^n E\left[e^{\alpha(X_i - E[X_i])}\right], \end{aligned} \tag{A.1}$$

where the equality follows from the independence of the X_i 's. In order to choose the right α , we need an upper bound for $E[e^{\alpha Y}]$ (where, in this case, $Y = X_i - E[X_i]$). Assume that Y is bounded, i.e., that there exist some $a, b \in \mathbf{R}$ such that $P\{a \leq Y \leq b\} = 1$; then, by convexity of the exponential function,

$$E\left[e^{\alpha(Y - E[Y])}\right] \leq \frac{b}{b-a} e^{\alpha a} - \frac{a}{b-a} e^{\alpha b} = e^{g(u)}$$

where $u = \alpha(b-a)$ and $g(u) = -pu + \log(1-p + pe^u)$ with $p = -a/(b-a)$. Is easy to verify that $g(0) = g'(0) = 0$ and that $g''(u) \leq 1/4$. Hence by Taylor's expansion, for a suitable $\theta \in \mathbf{R}$,

$$g(u) = g(0) + ug'(0) + \frac{u^2}{2} g''(\theta) \leq \frac{u^2}{8}$$

and we are able to conclude that

$$E\left[e^{\alpha Y}\right] \leq e^{\alpha E[Y] + \alpha^2(b-a)^2/8}.$$

We are ready to give a proof of Hoeffding's inequality. Let X_1, \dots, X_n be independent random variables such that $P\{X_i \in [a_i, b_i]\} = 1$ for $1 \leq i \leq n$; then, for every $\varepsilon > 0$, by applying equation (A.1) and taking $\alpha = 4\varepsilon / \sum_{i=1}^n (b_i - a_i)^2$, we have

$$P\left\{\sum_{i=1}^n X_i - E\left[\sum_{i=1}^n X_i\right] \geq \varepsilon\right\} \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

In a similar way, we can obtain that

$$P\left\{\sum_{i=1}^n X_i E\left(\sum_{i=1}^n X_i\right) \leq -\varepsilon\right\} \leq \exp\left(-\frac{2\varepsilon^2}{\sum_{i=1}^n (b_i - a_i)^2}\right).$$

We conclude this section by stating two lemmas that restate Hoeffding's inequality in a form which is used in some of the proofs of this thesis. Here, we use the notation $\Pr\{A\}$ to denote the probability of an event A whenever we do not want to formally specify the probability space over which the event itself is defined.

Lemma A.2.1 *Let X, X_1, \dots, X_n be independent, identically distributed, binomial random variables such that $\Pr\{X = 1\} = 1/2 + \delta$, for $1/2 < \delta < 1$. Then,*

$$\Pr\left\{\sum_{i=1}^n X_i \leq n/2\right\} < e^{-2n\delta^2}.$$

Proof . The statement follows by taking $b_i - a_i = 1$, for $1 \leq i \leq n$, and replacing ε by $n\delta$. ■

Lemma A.2.2 *Let X, X_1, \dots, X_n be independent and identically distributed random variables taking values in $[0, 1]$. Then, for every $\varepsilon > 0$,*

$$\Pr\left\{(1 - \varepsilon)E(X) \leq \sum_{i=1}^n X_i/n \leq (1 + \varepsilon)E(X)\right\} > 1 - 2e^{-2n(E(X)\varepsilon)^2}$$

Proof . The claim follows by taking $b_i - a_i = 1$, for $1 \leq i \leq n$, and replacing ε by $n\varepsilon E(X)$. ■

A.2.2 Improving randomized algorithms

As we have seen, randomized algorithms sometimes fail to give the correct answer and somehow signal this fact, let us say by giving an “undefined” output denoted by the special symbol \perp . A common strategy to decrease the probability of such an event, is to iterate the algorithm, usually for no more than a specified number of times, and to give as output the first non-undefined output, or some other function of the non-undefined outputs obtained in the iterations (or to output \perp if, for all the specified number of iterations, the algorithm always outputs \perp).

Here, we try to formalize this idea only with regard to probabilistic issues. For this reason, all the interpretations of the probabilistic statements of this subsection in terms of randomized algorithms is printed in a smaller font.

Let X, X_1, \dots, X_n be independent and identically distributed random variables taking values in a set $\mathbf{X} \subseteq \mathbf{R}$ and let $K \subseteq \mathbf{X}$ be a fixed measurable set.

Clearly, our idea is to use X to represent the output of a randomized algorithm and K to mean the set of its intended output. Usually, in this thesis, $K = \mathbf{N}$ and \mathbf{X} is as simple as $K \cup \{\perp\}$, where we assume to represent \perp in \mathbf{R} by some number not in K . Finally, the X_i 's represent the output of n iterations of the algorithm.

Let now Y, Y_1, \dots, Y_n be independent and identically distributed random variables taking values in \mathbf{X} such that

$$\Pr\{Y \in H\} = \Pr\{X \in H \mid X \in K\} \quad (\text{A.2})$$

for every measurable set $H \subseteq K$.

Again, our idea is to use Y to represent the intended output of the algorithm, together with its desired distribution. Equation (A.2), in fact, formally states that Y should be distributed like X whenever (conditionally to) X is different from \perp . The same, due to the identity of the distribution, is true for the Y_i 's.

We now show that, for every measurable function $F : K^n \rightarrow \mathbf{R}^m$ and every measurable set $M \in \mathbf{R}^m$, it holds that

$$\Pr\{F(X_1, \dots, X_n) \in M \mid X_1 \in K, \dots, X_n \in K\} = \Pr\{F(Y_1, \dots, Y_n) \in M\}. \quad (\text{A.3})$$

For the sake of brevity, let

$$F_i^{-1}(M) = \{x : F(y_1, \dots, y_{i-1}, x, y_{i+1}, \dots, y_n) \in M, y_1, \dots, y_n \in K\}$$

for $1 \leq i \leq n$. Then

$$\begin{aligned} \Pr\{F(X_1, \dots, X_n) \in M \mid X_1 \in K, \dots, X_n \in K\} &= \frac{\Pr\{\bigcap_{i=1}^n X_i \in F_i^{-1}(M) \cap \bigcap_{i=1}^n X_i \in K\}}{\Pr\{X \in K\}^n} \\ &= \prod_{i=1}^n \frac{\Pr\{X_i \in F_i^{-1}(M) \cap X_i \in K\}}{\Pr\{X \in K\}} \\ &= \prod_{i=1}^n \Pr\{X \in F_i^{-1}(M) \mid X \in K\} \\ &= \prod_{i=1}^n \Pr\{Y \in F_i^{-1}(M)\} \\ &= \Pr\{\bigcap_{i=1}^n Y_i \in F_i^{-1}(M)\} \\ &= \Pr\{F(Y_1, \dots, Y_n) \in M\} \end{aligned}$$

Equation (A.3) states that the probability of every event $F(X_1, \dots, X_n) \in M$ obtained taking some function F of the outputs of n iterations of a probabilistic algorithm, given that all the iterations are non-undefined, is equal to the probability of the same event $F(Y_1, \dots, Y_n) \in M$, expressed in terms of the intended outputs distribution.

Elementary case

Here we investigate the first strategy mentioned at the beginning of this section: to iterate the algorithm, giving in output the first non-undefined output so obtained.

Under the previous hypotheses, let $\tau_K = \inf\{t : 1 \leq t \leq n, X_t \in K\}$ (assuming, as usual, $\inf \emptyset = \infty$). Then, for every measurable set $H \subseteq K$, it holds that

- (i) $\Pr\{X_{\tau_K} \in H \mid \tau_K < \infty\} = \Pr\{X \in H \mid X \in K\} = \Pr\{Y \in H\},$
- (ii) $\Pr\{\tau_K = \infty\} = \Pr\{X \notin K\}^n.$

The second property is trivially verifiable; the first comes from

$$\begin{aligned}
 \Pr\{X_{\tau_K} \in H \mid \tau_K < \infty\} &= \frac{\Pr\{X_{\tau_K} \in H \cap \bigcup_{i=1}^n \tau_K = i\}}{\sum_{i=1}^n \Pr\{\tau_K = i\}} \\
 &= \frac{\sum_{i=1}^n \Pr\{X_i \in H \cap \tau_K = i\}}{\sum_{i=1}^n \Pr\{\tau_K = i\}} \\
 &= \frac{\sum_{i=1}^n \Pr\{\bigcap_{j=1}^{i-1} X_j \notin K \cap X_i \in K \cap X_i \in H\}}{\sum_{i=1}^n \Pr\{\bigcap_{j=1}^{i-1} X_j \notin K \cap X_i \in K\}} \\
 &= \frac{\Pr\{X \in H\} \sum_{i=1}^n \Pr\{\bigcap_{j=1}^{i-1} X_j \notin K\}}{\Pr\{X \in K\} \sum_{i=1}^n \Pr\{\bigcap_{j=1}^{i-1} X_j \notin K\}} \\
 &= \Pr\{X \in H \mid X \in K\}.
 \end{aligned}$$

Property (i) tells that if one of the iterations of the algorithm is ever successful, then that output has the same distribution as the intended one. Moreover, Property (ii) says that the probability that such “iteration strategy” has an undefined output decreases exponentially with the number of iterations.

A more complex situation

Now we assume that, after a certain number of iterations of a randomized algorithm, we want to return some function of the outputs of all the non-undefined iterations, such as, for instance, their median, or mean.

Under the previous hypotheses, given a family of \mathbf{R} valued measurable functions $\{f_n\}_{n>0}$, let $\{f_n^K\}_{n>0}$ be the corresponding family of functions defined as

$$f_n^K(x_1, \dots, x_n) = f_m(x_{i_1}, \dots, x_{i_k})$$

where $i_1 < \dots < i_k$ are exactly all the indexes for which $x_{i_m} \in K$ with $1 \leq k \leq m \leq n$, for every $n > 0$. Observe that the functions f_n^K are obviously all \mathbf{R} valued measurable functions, for $n > 0$.

Clearly, f_n^K is the function computed at the end of the n iterations on all the outputs, that, by definition, computes the function f_k (for instance, the mean, or the median) only on the k non-undefined outputs of all the iterations.

Let $\eta_K = |\{t : 1 \leq t \leq n, X_t \in K\}|$. Then for every measurable set $H \subseteq K$ and every $k > 0$, it holds that

$$\begin{aligned} \Pr\{f_n^K(X_1, \dots, X_n) \in H \mid \eta_K = k\} &= \Pr\{f_k(X_1, \dots, X_k) \in H \mid X_1 \in K, \dots, X_k \in K\} \\ &= \Pr\{f_k(Y_1, \dots, Y_k) \in H\}. \end{aligned}$$

This property formally states that the above mentioned strategy is “correct”, in the sense that the value of f_n^K on all outputs, given there were k non-undefined outputs, is distributed as f_k computed over k intended outputs of the algorithm.

The second equality comes from equation (A.3). To prove the first inequality, define, for the sake of brevity, the events

$$A(\Gamma) = \bigcap_{i \in \Gamma} X_i \in K \quad \text{e} \quad B(\Gamma) = \bigcap_{1 \leq i \leq n, i \notin \Gamma} X_i \notin K;$$

where $\Gamma = (i_1, \dots, i_k)$ is any ordered k -tuple of indexes without repetitions, i.e., $1 \leq i_1 < \dots < i_k \leq n$ and $1 \leq k \leq n$. Then,

$$\begin{aligned} \Pr\{f_n^K(X_1, \dots, X_n) \in H \mid \eta_K = k\} &= \frac{\Pr\{f_n^K(X_1, \dots, X_n) \in H \cap \bigcup_{\Gamma: |\Gamma|=k} A(\Gamma) \cap B(\Gamma)\}}{\binom{n}{k} \Pr\{X \in K\}^k \Pr\{X \notin K\}^{n-k}} \\ &= \frac{\sum_{\Gamma=(i_1, \dots, i_k)} \Pr\{f_k(X_{i_1}, \dots, X_{i_k}) \in H \cap A(\Gamma)\} \Pr\{B(\Gamma)\}}{\binom{n}{k} \Pr\{X \in K\}^k \Pr\{X \notin K\}^{n-k}} \\ &= \frac{\binom{n}{k} \Pr\{f_k(X_1, \dots, X_k) \in H \cap A(\{1, \dots, k\})\} \Pr\{X \notin K\}^{n-k}}{\binom{n}{k} \Pr\{X \in K\}^k \Pr\{X \notin K\}^{n-k}} \\ &= \frac{\Pr\{f_k(X_1, \dots, X_k) \in H \cap A(\{1, \dots, k\})\}}{\Pr\{X \in K\}^k} \\ &= \Pr\{f_k(X_1, \dots, X_k) \in H \mid X_1 \in K, \dots, X_k \in K\}. \end{aligned}$$

An upper bound

Usually, we would like to prove some concentration result about the output obtained applying f_n^K . To this aim, we assume to have an (exponential) upper bound on the probability of the concentration of f_k applied to k intended outputs of the algorithm. Hence, we show a corresponding (exponential) bound on the concentration of f_n^K .

Under the previous hypotheses, define $s_K = \Pr\{X \in K\}$ and, for a given measurable set $H \subseteq K$, assume that there exists a $0 < c_H < 1$ such that, for every $k > 0$,

$$\Pr\{f_k(Y_1, \dots, Y_k) \in H\} \leq e^{-kc_H}.$$

Then, it holds that

$$\Pr\{f_n^K(X_1, \dots, X_n) \in H \mid \eta_K > 0\} < \frac{1}{s_K} \left(1 - \frac{s_K c_H}{2}\right)^n. \quad (\text{A.4})$$

Observe, in fact, that

$$\begin{aligned}
\Pr\{f_n^K(X_1, \dots, X_n) \in H \mid \eta_K > 0\} &= \frac{\Pr\{f_n^K(X_1, \dots, X_n) \in H \cap \eta_K > 0\}}{\Pr\{\eta_K > 0\}} \\
&= \frac{\sum_{k=1}^n \Pr\{f_n^K(X_1, \dots, X_n) \in H \mid \eta_K = k\} \Pr\{\eta_K = k\}}{1 - (1 - \Pr\{X \in K\})^n} \\
&\leq \frac{\sum_{k=1}^n e^{-kc_H} \binom{n}{k} \Pr\{X \in K\}^k \Pr\{X \notin K\}^{n-k}}{1 - (1 - \Pr\{X \in K\})^n} \\
&< \frac{\sum_{k=0}^n \binom{n}{k} (e^{-c_H} \Pr\{X \in K\})^k \Pr\{X \notin K\}^{n-k}}{1 - (1 - \Pr\{X \in K\})^n} \\
&= \frac{(1 - \Pr\{X \in K\})(1 - e^{-c_H})^n}{1 - (1 - \Pr\{X \in K\})^n} \\
&\leq \frac{(1 - \Pr\{X \in K\})^{c_H/2}}{1 - (1 - \Pr\{X \in K\})}
\end{aligned}$$

where the last inequality follows from the facts that $1 - e^{-x} \geq x/2$ and $x^n \leq x$ whenever $x \in [0, 1]$ and $n > 0$.

A.2.3 The method of conditional expectation

We conclude this appendix with a detailed discussion, from the probabilistic viewpoint, of the so called *method of conditional expectation*. Let X_1, \dots, X_n be independent random variables (not necessarily identically distributed); moreover, to avoid boring technical details concerning the existence of the conditional expectation used in the following, assume that each X_i takes values in some discrete set¹ $\mathbf{X} \subset \mathbf{R}$, for $1 \leq i \leq n$. Finally, let $f : \mathbf{R}^n \rightarrow \mathbf{R}$ be some fixed function.

If we define Z as the random variable $Z(\omega) = f(X_1(\omega), \dots, X_n(\omega))$, then, by the *internality* of the expectation,

$$\inf_{\omega \in \Omega} Z(\omega) \leq E[Z] \leq \sup_{\omega \in \Omega} Z(\omega).$$

In our hypotheses, this means in particular that there exists some $\tilde{\omega} \in \Omega$ such that $E[Z] \leq Z(\tilde{\omega})$, or, that there exist $\tilde{x}_i = X_i(\tilde{\omega}) \in \mathbf{R}$, for $1 \leq i \leq n$, such that

$$E[f(X_1, \dots, X_n)] \leq f(\tilde{x}_1, \dots, \tilde{x}_n).$$

The aim of the method of conditional expectation is to obtain such \tilde{x}_i 's by computing some conditional expectations.

For the sake of brevity define, for $1 \leq k \leq n$, the functions $g_k : \mathbf{R}^k \rightarrow \mathbf{R}$ such that

$$g_k(x_1, \dots, x_k) = E[Z \mid X_1 = x_1, \dots, X_k = x_k].$$

By a straightforward application of the definition of conditional expectation and the elementary property of iterated expectations, one can verify that the following statements are true, for every $x_1, \dots, x_n \in \mathbf{R}$ and $1 \leq k < n$:

- (a) $E[Z] = E[g_1(X_1)]$,
- (b) $g_k(x_1, \dots, x_k) = E[g_{k+1}(x_1, \dots, x_k, X_{k+1})]$ and

¹this is usually stated by saying that the X_i 's are *simple* random variables.

$$(c) \quad g_n(x_1, \dots, x_n) = f(x_1, \dots, x_n),$$

$$(d) \quad E[Z] \leq \max_{x \in \mathbf{X}} g_1(x) \text{ and}$$

$$(e) \quad g_k(x_1, \dots, x_k) \leq \max_{x \in \mathbf{X}} g_{k+1}(x_1, \dots, x_k, x).$$

Hence, if the \hat{x}_i 's are recursively defined as

$$\hat{x}_i = \begin{cases} \arg \max_{x \in \mathbf{X}} g_1(x) & \text{if } i = 1, \\ \arg \max_{x \in \mathbf{X}} g_i(\hat{x}_1, \dots, \hat{x}_{i-1}, x) & \text{if } i > 1, \end{cases}$$

from (c)–(e), one can conclude that $E[Z] \leq f(\hat{x}_1, \dots, \hat{x}_n)$.

SYMBOLS AND ABBREVIATIONS

In this appendix we summarize some notation, together with some pointer to relevant literature for the notions used in this thesis that are not explicitly defined in other chapters.

Some Complexity and Approximation Classes

P languages recognizable in polynomial time by a (deterministic) Turing machine (Garey and Johnson, 1979).

FP functions computable in polynomial time by a (deterministic) Turing machine (Garey and Johnson, 1979).

NP languages recognizable in polynomial time by a nondeterministic Turing machine (Garey and Johnson, 1979).

#P functions yielding the number of accepting computations in nondeterministic polynomial time Turing machines (Valiant, 1979).

RP languages recognizable in probabilistic polynomial time with one-sided error (Balcázar et al., 1995).

NC^K languages recognized (functions computed by) log-space uniform boolean circuits of bounded fan-in, polynomial size and $O(\log^k n)$ depth (Karp and Ramachandran, 1990; Balcázar, Díaz, and Gabarró, 1990).

\mathcal{B} ($\mathcal{B}(s(n), i(n), d(n))$) class of languages recognized by Turing machines with simultaneous complexity bounds (Section 2.2).

NPO NP optimization problems (Section 6.1).

GLO NPO problems with guaranteed local optima (Section 6.2.2).

EG (and RS-good, RS-derandomizable) expectation-guaranteed NPO problems and related subclasses (Section 6.3).

APX NPO problems approximable in polynomial time within a constant factor (Section 6.1.2).

PTAS NPO problems admitting a polynomial time approximation scheme (Section 6.1.2).

FPTAS NPO problems admitting a fully polynomial time approximation scheme (Section 6.1.2).

MaxNP (and MaxSNP) logically defined NPO problems (Section 6.2.2).

Models of Computation

RAM random access machine (Section 1.1.1).

PrRAM probabilistic RAM (Section 1.1.2)

TM (NTM) (nondeterministic) Turing machine (Section 2.2).

1-AuxPDA (1-NAuxPDA, 1-UAuxPDA) (nondeterministic, unambibuous) one-way auxiliary push-down automata (Section 2.3).

Number Rings and Fields

\mathbf{N} , \mathbf{Z} , \mathbf{Q} and \mathbf{R} denote, respectively, the ring of *integer* and *natural* number and the field of *rational* and *real* numbers.

Basic notation

$|\omega|$ the lenght of the string ω

$\#A$ the cardinality of set A

LIST OF ALGORITHMS

1.1	a uniform random integer numbers generator.	8
3.1	An algorithm to increase the success probability of a u.r.g..	30
3.2	An algorithm to increase the probability of correct approximation of a r.a.s.. . .	32
3.3	a uniform random generator for regular languages.	34
3.4	An u.r.g. for (ambiguously) described combinatorial structures.	35
3.5	A r.a.s. for ambiguously described combinatorial structures.	37
3.6	A u.r.g. for the product of combinatorial structures.	39
3.7	A u.r.g. for the union of combinatorial structures.	40
4.1	a uniform random generator of derivation trees.	43
4.2	Part of Earley's algorithm modified to count derivation trees.	46
4.3	Counting derivation trees.	48
6.1	Local Search: schema of algorithm.	65
6.2	RandomSearch.	66
6.3	Conditional expectation.	70
6.4	An approximation algorithm for EG problems.	71
A.1	Computing the bit size of an integer.	82

LIST OF FIGURES

1.1	A random access machine.	2
1.2	A PrRAM.	6
2.1	A Turing machine.	20
2.2	An extended Turing machine for recognizing relations.	22
2.3	An auxiliary pushdown automaton.	24
2.4	An extended auxiliary pushdown automaton for recognizing relations.	26
3.1	An Example of Ambiguous description.	34
5.1	Some classes of p -relations.	58

LIST OF TABLES

1.1	Meaning of RAM instructions.	3
1.2	Logarithmic cost of RAM instructions.	4
1.3	Meaning of PrRAM instructions.	6
1.4	Cost of PrRAM instructions.	7

BIBLIOGRAPHY

- A. V. Aho and J. D. Ullman. *The Theory of Parsing, Translation and Compiling - Vol.I: Parsing*. Prentice Hall, Englewood Cliffs, NJ, 1972.
- A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
- P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. *Information Processing Letters*, 57:151–158, 1996.
- E. Allender, D. Bruschi, and G. Pighizzini. The complexity of computing maximal word functions. *Computational Complexity*, 3:368–391, 1993.
- L. Alonso. Uniform generation of a Motzkin word. *Theoretical Computer Science*, 134(2): 529–536, Nov. 1994.
- G. Ausiello and M. Protasi. Local search, reducibility and approximability of NP-optimization problems. *Information Processing Letters*, 54:73–79, 1995.
- A. Avellone and M. Goldwurm. Analysis of algorithms for the recognition of rational and context-free trace languages. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 32(4-5-6):141–152, 1998.
- J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity II*. Springer, Berlin, 1990.
- J. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I*. Springer, Berlin, 1995.
- E. Barcucci, R. Pinzani, and R. Sprugnoli. Génération aléatoire de chemins sous-diagonaux. In P. Leroux and C. Reutenauer, editors, *Actes du 4ème Colloque Séries Formelles et Combinatoire Algébrique*, Monreal, 1992.
- J. Barwise. An introduction to first-order logic. In *Handbook of Mathematical Logic*. North Holland, 1977.
- A. Bertoni and M. Goldwurm. On ranking 1-way finitely ambiguous NL languages and $\sharp P_1$ -complete census functions. *RAIRO Informatique théorique et Applications/Theoretical Informatics and Applications*, 27(2):135–148, 1993.
- A. Bertoni, G. Mauri, and N. Sabadini. A hierarchy of regular trace languages and some combinatorial applications. In A. Ballester, D. Cardus, and E. Trillas, editors, *Proceedings of the 2nd World Conf. on Mathematics at the Service of Man, Las Palmas (Canary Island) Spain*, pages 146–153. Universidad Politecnica de Las Palmas, 1982.

- A. Bertoni, G. Mauri, and N. Sabadini. Unambiguous regular trace languages. In G. Demetrescu, J. Katona, and A. Salomaa, editors, *Proceedings of the Coll. on Algebra, Combinatorics and Logic in Computer Science*, volume 42 of *Colloquia Mathematica Soc. J. Bolyai*, pages 113–123. North Holland, Amsterdam, 1985.
- A. Bertoni, G. Mauri, and N. Sabadini. Membership problems for regular and context free trace languages. *Information and Computation*, 82:135–150, 1989.
- A. Bertoni, D. Bruschi, and M. Goldwurm. Ranking and formal power series. *Theoretical Computer Science*, 79(1):25–35, Feb. 1991a.
- A. Bertoni, M. Goldwurm, and N. Sabadini. The complexity of computing the number of strings of given length in context-free languages. *Theoretical Computer Science*, 86(2):325–342, 1991b.
- A. Bertoni, C. Mereghetti, and G. Pighizzini. On languages accepted with simultaneous complexity bounds and their ranking problem. *Lecture Notes in Computer Science*, 841, 1994.
- A. Bertoni, M. Goldwurm, and M. Santini. Random generation and approximate counting of ambiguously described combinatorial structures. Technical Report RI-DSI 236-99, Dipartimento di Scienze dell'Informazione, Milano, 1999.
- A. Bertoni, M. Goldwurm, and M. Santini. Random generation and approximate counting of ambiguously described combinatorial structures. In H. Reichel and S. Tison, editors, *Proceedings of 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, number 1770 in *Lecture Notes in Computer Science*, pages 567–580. Springer, 2000.
- P. Billingsley. *Probability and Measure*. John Wiley & Sons, Inc., New York, 1979.
- D. P. Bovet and P. Crescenzi. *Introduction to the Theory of Complexity*. Prentice Hall, 1993.
- F.-J. Brandenburg. On one-way auxiliary pushdown automata. In H. W. H. Tzschach and H. K.-G. Walter, editors, *Proceedings of the 3rd GI Conference on Theoretical Computer Science*, volume 48 of *Lecture Notes in Computer Science*, pages 132–144, Darmstadt, FRG, Mar. 1977. Springer.
- L. Breveglieri, A. Cherubini, and S. Crespi-Reghezzi. Real-time scheduling by queue automata. In J. Vyttopil, editor, *Proceedings of Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 571 of *LNCS*, pages 131–148, Berlin, Germany, Jan. 1991. Springer.
- L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi-Reghezzi. Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, 7, 1996.
- G. Buntrock and K. Lorys. On growing context-sensitive languages. In W. Kuich, editor, *Automata, Languages and Programming, 19th International Colloquium*, volume 623 of *Lecture Notes in Computer Science*, pages 77–88, Vienna, Austria, 13–17 July 1992. Springer-Verlag.
- H. Chernoff. A measure of asymptotic efficiency of tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.
- C. Choffrut and M. Goldwurm. Rational transductions and complexity of counting problems. *Mathematical Systems Theory*, 28(5):437–450, 1995.
- N. Chomsky and M.-P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North-Holland, Amsterdam, The Netherlands, 1963.

- L. Comtet. Calcul pratique des coefficients de Taylor d'une fonction algébrique. *L'Enseignement Mathématique*, 10:267–270, 1964.
- S. A. Cook. Path systems and language recognition. In *Conference Record of Second Annual ACM Symposium on Theory of Computing*, pages 70–72, Northampton, Massachusetts, 4–6 May 1970.
- S. A. Cook. Characterizations of pushdown machines in terms of time-bounded computers. *Journal of the ACM*, 18(1):4–18, Jan. 1971.
- S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.
- P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical report, Nada Institute, Stockholm, Sweden, <http://nada.kth.se/viggo/compendium>, 1998.
- P. Crescenzi, V. Kann, M. Protasi, and L. Trevisan. Structure in approximation classes. In *Proceedings of the 1st Combinatorics and Computing Conference*, volume 959 of *Lectures Notes in Computer Science*, pages 539–548. Springer-Verlag, 1995.
- M. Davis. *Computability and Unsolvability*. McGraw-Hill, New York, 1958.
- M.-P. Delest and G. Viennot. Algebraic languages and polyomino enumeration. *Theoretical Computer Science*, 34(1-2):169–206, Nov. 1984.
- V. Diekert and Y. Métivier. Partial commutation and traces. In G. Rozenberg and A. Salomaa, editors, *Handbook on Formal Languages*, volume III, pages 457–527. Springer, Berlin-Heidelberg, 1997.
- V. Diekert and G. Rozenberg. *The Book of Traces*. World Scientific, Singapore, 1995.
- J. Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, Feb. 1970.
- S. Eilenberg and M. P. Schützenberger. Rational sets in commutative monoids. *Journal of Algebra*, 13:173–191, 1969.
- P. Erdős and J. Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.
- R. Fagin. Generalized first-order spectra and polynomial-time recognizable sets. In R. M. Karp, editor, *Complexity of Computation: Proceedings of a Symposium in Applied Mathematics of the American Mathematical Society and the Society for Industrial and Applied Mathematics*, pages 43–73. American Mathematics Society, Providence R.I., 1974.
- D. Feldman, R. Impagliazzo, M. Naor, N. Nisan, S. Rudich, and A. Shamir. On dice and coins: Models of computation for random generation. *Information and Computation*, 104(2):159–174, June 1993.
- J. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, Inc., New York, 1968.
- P. Flajolet. Mathematical methods in the analysis of algorithms and data structures. In E. Börger, editor, *Trends in Theoretical Computer Science*, chapter 6, pages 225–304. Computer Science Press, Rockville, Maryland, 1988.

- P. Flajolet, P. Zimmerman, and B. Van Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1-2):1–35, 1994.
- M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., San Francisco, 1979.
- J. Gill. Computational complexity of probabilistic turing machines. *SIAM Journal of Computing*, 6:675–695, 1977.
- A. V. Goldberg and M. Sipser. Compression and ranking. *SIAM Journal on Computing*, 20(3):524–536, June 1991.
- M. Goldwurm. Random generation of words in an algebraic language in linear binary space. *Information Processing Letters*, 54(4):229–233, 1995.
- D. H. Greene and D. E. Knuth. *Mathematics for the analysis of algorithms*, volume 1. Progress in computer science, Birkhäuser, Basel, CH, 1981.
- G. Grossi. *Expectation-Guaranteed Local Search and related Neural and Genetic Heuristics*. PhD thesis, Dipartimento di Scienze dell’Informazione – Università degli Studi di Milano, 1999.
- M. A. Harrison. *Introduction to Formal Language Theory*. Addison-Wesley, Reading, MA, 1978.
- D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing, Boston, 1997.
- W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58:13–30, 1963.
- J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Language, and Computation*. Addison-Wesley, Reading, MA, 1979.
- D. T. Huynh. The complexity of ranking simple languages. *Mathematical Systems Theory*, 23(1):1–19, 1990.
- D. T. Huynh. Effective entropies and data compression. *Information and Computation*, 90(1):67–85, Jan. 1991.
- M. Jerrum and A. Sinclair. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82:93–133, 1989.
- M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science*, 43(2-3):169–188, 1986.
- D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37:79–100, 1988.
- R. Karp and V. Ramachandran. A survey of parallel algorithms for shared-memory machines. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 869–941. North-Holland, 1990.
- R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

- R. M. Karp, M. Luby, and N. Madras. Monte-carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10:429–448, 1989.
- S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proceedings of the 35rd Annual IEEE Symposium on the Foundations of Computer Science*, pages 819–830. IEEE Computer Society, 1994.
- P. G. Kolaitis and M. N. Thakur. Logical definability of NP optimization problems. *Information and Computation*, 115:321–353, 1994.
- S. Lang. *Algebra*. Addison-Wesley, Reading, MA, 3rd edition, 1994.
- A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms*. Academic Press, 2nd edition, 1978.
- C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.
- C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *Journal of Computer and System Science*, 43:425–440, 1991.
- H. J. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, New York, 1967.
- J. Sakarovitch. On regular trace languages. *Theoretical Computer Science*, 52(1-2):59–75, 1987.
- C. P. Schnorr. Optimal algorithms for self-reducible problems. In S. Michaelson and R. Milner, editors, *Third International Colloquium on Automata, Languages and Programming*, pages 322–337, University of Edinburgh, 20–23 July 1976. Edinburgh University Press.
- J. Simon. On the difference between one and many. In A. Salomaa and M. Steinby, editors, *Proceedings of the 4th Colloquium on Automata, Languages and Programming*, volume 52 of *LNCS*, pages 450–491, Turku, Finland, July 1977. Springer.
- A. Sinclair. *Randomized Algorithms for Counting and Generating Combinatorial Structures*. PhD thesis, University of Edinburgh, 1988.
- L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, Oct. 1977.
- L. J. Stockmeyer. The complexity of approximate counting. In *ACM Symposium on Theory of Computing*, pages 118–126, Baltimore, USA, Apr. 1983. ACM Press.
- A. Szepietowski. *Turing Machines with Sublogarithmic Space*. Lecture Notes in Computer Science 843. Springer Verlag, 1994.
- L. G. Valiant. The complexity of combinatorial computations: An introduction. *GI Jahrestagung Informatik*, pages 326–337, 1978.
- L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, Apr. 1979.
- L. G. Valiant, S. Skyum, S. J. Berkowitz, and C. Rackoff. Fast parallel computation of polynomials using few processors. *SIAM Journal on Computing*, 12(4):641–644, Nov. 1983.
- J. von zur Gathen and G. Seroussi. Boolean circuits versus arithmetic circuits. *Information and Computation*, 91(1):142–154, Mar. 1991.

INDEX

- arithmetic circuit
 - constants, 73
 - depth, 73
 - inputs, 73
 - output, 73
 - representing a polynomial, 74
 - exactly, 74
 - size, 73
- one-way auxiliary pushdown automata
 - accepting state, 24
 - ambiguity, 25
 - auxiliary tape, 24
 - computational complexity, 25
 - finite control, 24
 - initial pushdown symbol, 24
 - initial state, 24
 - input tape, 24
 - language accepted, 24
 - next-move function, 24
 - nondeterministic, 25
 - polynomially ambiguous, 49
 - pushdown store, 24
 - realizable pair, 49
 - semantic, 24
 - space, 25
 - states, 24
 - surface configuration, 49
 - symbol, 24
 - time, 25
 - unambiguous, 25
- context-free grammar
 - ambiguity, 42
 - Chomsky normal form, 42
 - finitely ambiguous, 42
 - polynomially ambiguous, 42
 - unambiguous, 42
- context-free language
 - inherently ambiguous, 47
 - polynomially ambiguous, 47
 - unambiguous, 47
- combinatorial structure
 - census function, 29
 - description, 33
 - domain, 29
 - product, 39
 - randomized approximation scheme, 31
 - randomized exact counter, 32
 - size, 29
 - uniform random generator, 30
 - union, 39
- local search
 - global maximum, 65
 - initial solution, 65
 - local maximum, 65
- NP optimization problem, 60
 - AP-reduction, 64
 - approximation algorithm, 61
 - approximation scheme, 61
 - fully polynomial, 61
 - polynomial, 61
 - approximation-preserving reductions, 63
 - closure, 64
 - complete, 64
 - decision version, 60
 - expectation-guaranteed, 68
 - feasible solutions, 60
 - guaranteed local optima, 65
 - hard, 64
 - instances, 60
 - logical definability, 62
 - measure, 60
 - objective function, 60
 - optimum, 60
 - optimum solution, 60
 - performance ratio, 61
 - polynomially bounded, 60

- relative error, 61
- value, 60
- neighborhood structure
 - h -bounded, 64
 - neighbor, 64
 - neighborhood, 64
- polynomial, 72
 - degree, 72
 - evaluation, 72
 - indeterminate, 72
 - substitution, 72
- probabilistic random access machine
 - computational complexity, 7
 - number of random bits, 7
 - random tape, 6
 - semantic, 6
- random access machine
 - computational complexity, 4
 - indirect addressing, 2
 - input tape, 2
 - instruction pointer, 3
 - memory, 2
 - memory map, 3
 - output tape, 2
 - program, 2
 - semantic, 3
 - value of an operand, 3
- Turing machine
 - accepting state, 20
 - ambiguity, 21
 - finite control, 20
 - initial state, 20
 - input tape, 20
 - inversions, 21
 - language accepted, 21
 - next-move function, 20
 - nondeterministic, 21
 - oracle tape, 13
 - output tape, 20
 - query state, 13
 - semantic, 20, 21
 - space, 21
 - states, 20
 - tape symbol, 20
 - time, 21
- trace language, 52
 - ambiguity, 52
 - finitely ambiguous, 52
 - polynomially ambiguous, 52
 - rational, 52
- extended one-way auxiliary pushdown automata
 - ambiguity, 26
 - relation accepted, 26
 - space, 26
 - time, 26
- extended Turing machine
 - ambiguity, 21
 - inversions, 21
 - relation accepted, 21
 - semantic, 21
 - space, 21
 - time, 21
- $p@p$ -relation
 - almost uniform generator, 11
 - tolerance, 11
 - exact uniform generator, 12
 - randomized approximate counter, 12
 - randomized approximation scheme, 12
- ambiguity
 - one-way auxiliary pushdown automata, 25, 49
 - extended, 26
 - Turing machine, 21
 - extended, 21
 - context-free
 - grammar, 42
 - language, 47
 - description, of a combinatorial structure, 34
 - trace language, 52
- ambiguous description, *see* combinatorial structure
- arithmetic circuit, 73
- bounded expected time probabilistic algorithms, 5
- bounded time probabilistic algorithms, 5
- combinatorial optimization, 60
- combinatorial structure, 29
- computational complexity
 - one-way auxiliary pushdown automata
 - extended, 26
 - nondeterministic, 25
 - random access machine, 4
 - probabilistic, 7
 - Turing machine, 21

- extended, 21
 - nondeterministic, 21
- context-free grammar, 42
- cost criterion
 - random access machine, 4
 - logarithmic, 4
 - uniform, 4
- description, *see* combinatorial structure
- deterministic finite automaton, 33
- dotted productions, *see* weighted dotted productions
- expectation-guaranteed, *see* NP optimization problem
- extended one-way auxiliary pushdown automata, 25
- extended Turing machine, 21
- feasible model of computation, 6
- finitely ambiguous
 - context-free language, 47
 - trace language, 52
- language accepted
 - one-way auxiliary pushdown automata, 24
 - Turing machine, 21
- lexicographic order, 18
- local search paradigm, 64
- method of conditional expectation, 69
- model of computation, 1
- neighborhood structure, 64
- nondeterministic one-way auxiliary pushdown automata, *see* one-way auxiliary pushdown automata
- nondeterministic Turing machine, *see* Turing machine
- one-way auxiliary pushdown automata
 - extended, *see* extended one-way auxiliary pushdown automata
- one-way auxiliary pushdown automaton, 24
- one-way unambiguous auxiliary pushdown automata, *see* one-way auxiliary pushdown automata
- p -relation, 10
 - self-reducible, 10
- partial recursive functions, 3
- permanent, 74
- polynomial hierarchy, 13
- polynomially ambiguous
 - one-way auxiliary pushdown automata, 49
 - context-free language, 47
 - trace language, 52
- polynomially related, 5
- probabilistic random access machine, 6
- probabilistic Turing machine, 5
- random access machine, 2
- randomized approximation scheme
 - combinatorial structure, 31
 - $p@p$ -relation, 12
- randomized counter
 - combinatorial structure (exact), 32
 - $p@p$ -relation (approximate), 12
- rank
 - for formal languages, 18
 - for relations, 18
- relation accepted
 - extended one-way auxiliary pushdown automata, 26
 - extended Turing machine, 21
- search functions, 9
- semantic
 - one-way auxiliary pushdown automata, 24
 - extended, 25
 - random access machine, 3
 - probabilistic, 6
 - Turing machine, 20
 - extended, 21
 - nondeterministic, 21
- slice relation, 11, 19
- source of randomness, 5
- string relations, 9
- Turing machine, 2
 - deterministic, 20
 - extended, *see* extended Turing machine
 - with oracle, 13
- uniform generator
 - combinatorial structure, 30
 - $p@p$ -relation (almost), 11
 - $p@p$ -relation (exact), 12
- unrank
 - for formal languages, 18

- for relations, 18
- unranking
 - for formal languages, 19
 - for relations, 19
- versions of combinatorial problem, 9
- weighted dotted productions, 44

This thesis was written using *free* and *open source* software only. In particular,

- (i) Linux, the operating system, mainly due to *Linus Torvalds*;
- (ii) \TeX , together with \LaTeX and $\mathcal{A}\mathcal{M}\mathcal{S}\text{-}\text{\LaTeX}$, the typesetting software, respectively and mainly due to *Donald Knuth*, *Leslie Lamport* and the *American Mathematical Society*;
- (iii) $\{\text{X}\}$ Emacs, together with AUCTeX, RefTeX and XFig, the typesetting editor and environment, respectively and mainly due to *Richard Stallmann*, *Kresten Krab Thorup*, *Carsten Dominik* and *Supoj Sutanthavibul*.

I would like to thank all of them and the people of the free and open source community for giving us such an incredible effort in developing extraordinary pieces of software.